Hamming Distance as a Metric for the Detection of Side Channel in 802.11 Wireless Communications

by

Visal Chea

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Masters of Science

in

The Faculty of Business and Information Technology
and the program of Computer Science

University of Ontario Institute of Technology

Supervisor(s): Dr. Miguel Vargas Martin and Dr. Ramiro Liscano

April, 2015

Abstract

A wireless network can be exploited in many ways. One way is through intentionally corrupting the Frame Check Sequence (FCS) field by using a different Cyclic Redundancy Check (CRC) polynomial in order to create a side channel. Malicious nodes exploit the fact that normal unsuspecting nodes will immediately drop erroneous frames. A metric called the Hamming Distance (HD) was proposed for detection which distinguishes legitimate from illegitimate errors. The idea is to apply this HD measure to compare CRC values that are generated by different CRC polynomials. The hypothesis is that the average HD between two CRC values generated by two different CRC polynomial would be significantly far apart than those that are generated by the same CRC polynomial. The results show that this HD metric is effective in detecting the presence of side channel frames.

To the wonderful people in my life

To My Parents Sombath and Savi Chea,

My Brother Sydara Chea and

My Better Half Kelsey Anstey

For all the Love, Support, Strength and Encouragement to help me tackle all the challenges in order to succeed

Acknowledgments

This project was supported in part by the Natural Sciences and Engineering Research Council of Canada

and by

Defence Research & Development Canada.

I would like to thank my Supervisors Dr. Miguel Vargas Martin and Dr. Ramiro Liscano for their support, inspiration, mentorship and expert guidance. Both of you have helped me so much in sculpting me not only as a researcher but also as a person.

I would like to thank Mazda Salmanian, Ming Li and Peter Mason at Defence Research & Development Canada for all their insight and expert knowledge on this topic.

I would also like to thank my teammate Brent Moore for all the help and support while working on this project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

While wireless solutions allow for maximum portability and communication, it also presents many security risks that are aimed at exploiting this medium. The risk becomes more imperative when it relates to the mobile communication amongst the soldiers of Canadian Armed Forces. When the information relayed amongst soldiers could be classified or life critical, any exploitation is not acceptable. Knowing the types of exploitation is important when tasked in trying to secure communication channels between soldiers. One possible form of exploitation is done through manipulating the error detecting mechanism put in place to guard against the unreliable nature of wireless communication. Errors in the transmission of frames can occur from signal fading, collisions and shadowing effects which can be attributed to many factors such as terrain, distance of nodes and interference. To mitigate these unreliability issues, a Frame Check Sequence (FCS) field is added to the end of the frame which stores a Cyclic Redundancy Check (CRC) value calculated based on the contents of the frame. This value is calculated using a predetermined CRC polynomial before it is sent and checked at the receiver for consistency. An attack that may exploit this is to intentionally corrupt the FCS by means of choosing a different CRC polynomial in order to establish a covert communication channel for the exchange of information between malicious nodes. This type of attack is termed as a side channel communication. Other malicious nodes can now communicate through this side channel by interpreting these intentionally corrupted frames using the

same CRC polynomial. The surrounding nodes that are unaware of the use of a different CRC polynomial will interpret any frames from the malicious nodes as corrupted and thus making detection very difficult for this type of anomaly as it is seen as added noise.

1.1 Problem Statement

The amount of noise in a network is never constant which poses the first challenge in detection. If it were constant then it would be easy to detect the presence of side channel when there is an abnormal increase in corrupted frames. This is not the case because a baseline noise value cannot be determined for comparison due to the unpredictable nature of wireless communications. This is evident when a nearby microwave is turned on and all of a sudden the transmission rates between nodes come to almost a crawling stop due to a significant corruption of frames.

The next challenge of detection which stems from the type of network infrastructure that these side channels can coexist. These infrastructures usually lack a central monitor which increases the likelihood of errors when the communication medium is shared amongst its nodes. One infrastructure that possesses these characteristics and is of main interest in the field of side channel communication, is the Mobile Ad Hoc Networks (MANETs). In this Mobile Ad Hoc environment detection becomes very hard when the only metric used is the frame error rate caused by noisy environments. It is very difficult to distinguish normal noise from intentionally corrupted frames that appear like noise. This calls on the need to find another metric that can be used to identify an intentionally corrupted frame from a naturally corrupted frame. If a metric is found the next question

that arises is how it can be validated. Once validated, how will this metric be used? In other words will it be used as an input into a well-known detection algorithm or will there be an effective threshold that can be determined. This becomes the main focus of this thesis.

1.2 Contributions

This thesis's main contribution is to present the Hamming distance measure as a metric used for the detection of side channel communication. It also contributes a testing environment that could be used for any future work on the topic of side channel. As a consequence of the work in the testing environment, it also contributes to the existing literature on AirPcap and Wireshark with added knowledge on how to extract and format the data for custom use in MATLAB/Simulink simulation. Not only will this benefit any future work in this area but it will be very useful for any work done related to captured frame data. Another contribution is the insight into the use of the QualNet simulator that can be used as a starting point into future works.

1.3 Structure of Thesis

Chapter two provides background information on the MANET environment and the Cyclic Redundancy Check concept and also includes a survey of other types of side channels that are currently in the literature in hopes to give both similarities and more importantly the uniqueness of this thesis's side channel problem. It also includes a survey of general detection techniques used. The third chapter gives some insight into the

process in discovering the Hamming distance as a metric for use in detection which includes some preliminary work in MATLAB/Simulink. The fourth chapter explores possible implementation of a test environment in both hardware and software. This includes looking into possible modification of existing network cards and also an evaluation of simulators such as NS-2, QualNet, Sinalgo and MATLAB/Simulink. Chapter four concludes with the introduction of a hybrid approach where real data captures are incorporated into simulation. This is done with the combination of the AirPcap network adapter, Wireshark and MATLAB/Simulink. The fifth chapter describes use of the hybrid approach introduced as the testing environment in the methodology in validating and evaluating the effectiveness of the Hamming distance as a metric for detection of side channel communication. This entails a detailed overview of the experimental setup and its parameters, the use of the Perceptron Learning Algorithm to validate that the Hamming distance is an effective feature able to distinguish between intentionally corrupted frames from naturally corrupted ones. Chapter five concludes by presenting possible scores that can be used to evaluate and measure the effectiveness of the Hamming distance metric. Chapter six explores the possibility of finding a generalizable effective Hamming distance threshold value that can be used to detect the presence of side channel. The final chapter concludes the thesis by quickly revisiting the contributions and confirming that they have been delivered. This also contains a critical review of the thesis for possible weakness that may suggest future works to supplement or further the research in this domain.

Chapter 2

Background and Related Works

The purpose of this chapter is to give the reader a better understanding of the components that make up the problem and its environment. It allows a more detailed inspection into the components for easy comprehension of how they can be exploited even though some of them are not implemented in this thesis for testing. In particular the mobile and routing components are not implemented in testing. The reasoning for this will be offered later in Chapter four when the testing environment is explored. However, it is still advantages to offer a comprehensive review of these components in order to gain an understanding of the requirements that make up a real life side channel scenario for any future work. It also brings to light the feasibility of such exploitations when the environment is well known. The background section will start off with exploring the characteristics of MANETs, routing protocols and then move on to the concept of the Cyclic Redundancy Check (CRC). The related works section that follows gives the reader a clear distinction between this work's side channel problem and other works.

2.1 Background – Mobile Ad Hoc Networks

Mobile Ad Hoc Networks (MANETs) consists of autonomous mobile nodes that have no fixed infrastructure and no centralized control [1]. This autonomous characteristic allows these nodes the freedom to connect and communicate with neighbouring nodes in radio

range on the fly [2]. These nodes can form a network without any type of base station registration, alluding to the fact that MANETs lack a static infrastructure with a centralized controller. Without a central controller these autonomous nodes must act as both the router and host [1]. This ability for MANETs to setup and tear down on the fly is an advantage when there is no existing infrastructure but also a disadvantage that exposes the network to some vulnerabilities. Roy [1] also identifies that the mobility model of MANETs are hard to be reproduce and that the topology is always changing and may degrade the performance which warrants the high possibility of errors. The limited capabilities of a node's wireless radio only being able to either send or receive at one given time coupled with the challenges of sharing the communication medium also becomes an issue [1]. In order to mitigate the number of errors in transmission through overcoming the challenging physical aspects of MANETs, protocols must be put into place. Generic to all wireless protocols, MANETs also have the physical (PHY) and Data Link layers protocols in place to detect corrupted frames and limit the number of errors [3]. These errors are measured in bit error rates or frame error rates [3]. Included in the Data Link layer is the Media Access Control (MAC) Layer. The MAC layer is responsible for coordinating the sharing of the communication channel and implements error detecting mechanism that appends Frame Check Sequence (FCS) to any outgoing frame while the PHY layer establishes the connection [3]. The FCS is derived from applying the CRC generator function on the frame payload [3]. Once the CRC is calculated and appended to the FCS, the integrity of the frame sent can be evaluated by calculating the CRC on the frame payload at the receiving end. The CRC is calculated using the remainder from the division of the frame payload with a CRC generator function [3]. The

CRC concept will be reviewed in more detail later. Another important facet to look into is the type of routing protocol used for MANETs, as the topology is frequently changing. This will help to answer how nodes in a MANET are able to join and leave dynamically without any type of base station registration. It will also help to define the routing protocol used for future testing and implementation.

The dynamic topology nature of MANETs coupled with its limited resources, poses a challenge in finding an efficient and reliable routing protocol that will cater to it. The reason being is that existing routing protocols that can be found in Ethernet based LANS are sufficient to perform routing in MANETS but Kumaar et al. [4] warn that they may be vulnerable to attacks. These attacks are due to the fact that MANETs have no secure boundaries and thus allow the ability for compromised nodes to threaten the integrity of a network with no centralized management and limited power supply [4]. Abolhasan et al. [5] describes two types of routing algorithm that exist for current LANs which are link-state and distance-vector. Both of these routing algorithms have periodical updates and consume large amounts of bandwidth to keep track of these tables [5]. This is not ideal for MANETs as they are limited to resources such as power and memory space. Abolhasan et al. [5] then identify three other types that are more scalable to MANETS that are global/proactive, on-demand/reactive and hybrid routing algorithms. The difference between proactive and reactive is determined by when the routes to all destinations are requested. Proactive routing converges at start up, while reactive only request routes when needed by the source node. Hybrid routing is the combination of both proactive and reactive. Based on these routing algorithms there a many protocols

developed and implemented in different network environments. The one that is appropriate for the MANETs is the Optimized Link State Routing Protocol (OLSR) that combines the link-state information storage scheme and the route update efficiency of proactive routing algorithm [5]. The efficiency of the proactive routing relies on limiting the number of route broadcast by only updating the link-state information when there is a topology changes. OLSR does this by electing certain nodes for rebroadcast duty when a topology change has occurred [5]. This limits the use of available bandwidth. Understanding the routing behaviour of MANETs is important in order to account for possible unknown variables that could be overlooked during future testing.

2.2 Background – Cyclic Redundancy Checks

The CRC bits that are added to the FCS field of a frame are the result of the remainder when dividing the data payload polynomial with the CRC polynomial. For example, if the CRC polynomial was $x^3 + x + 1$ and the data payload was $x^7 + x^4 + x^2 + x$. The CRC polynomial becomes the divisor and the payload becomes the dividend. Both these polynomials are converted to binary format first before binary division is performed. Since the CRC value is the remainder, instead of dividing both these binary values a modulo operation can be performed instead.

Modulo Operation example:

1001011(Payload) modulo *1011(CRC Polynomial)* = *1001 (CRC Value)*

This CRC value is then appended to the end of the frame before it is transmitted. At the receiver side the remainder is calculated in the same way and compared with the CRC that was appended from the sender. Normally in hardware this is done on a very low level using feedback shift registers [6]. In the hardware implementation the data must first be serialized and process bit-by-bit per clock cycle. When software solutions that tried to mimic this bit-by-bit processing, it was found to be very slow and it consumed too many resources. Sarwate et al. [6] describe a method of implementing the CRC code in software. The method takes advantage of the ability of computers to handle pre-computed blocks of bits and store them into look-up tables. The idea comes from the known fact that the remainder of a given divisor and dividend is always going to be the same [6]. In other words, the remainder for all combinations of blocks bits and CRC polynomial bits are pre-computed and stored in the lookup table. This saves time during runtime because as the block of bits are encountered in the message payload, only a lookup is needed to find its corresponding remainder value. The question now to ask is whether or not the CRC polynomial makes a difference in error detection. The answer to that question is that not all CRC polynomials are created equal.

Koopman and Chakravarty [7] try to shed some light on the effectiveness of detecting errors in using different CRC polynomials. Even though the use of these CRC codes have helped to detect transmission error, there is no guarantee that during transmission certain bits have been flipped in such a combination that at the receiver end the calculation produces an identical CRC sent [7]. They also suggest that a well-known CRC polynomial used for calculation in one application may not be as effective in

another. This suggest that the effectiveness can be measured and are different for every CRC polynomial. They identify that measure of how effective a CRC polynomial is, can be calculated by finding the Hamming distance which measures the minimum number of bit flips required to make an error undetectable [7]. Due to the nature of wireless transmission it is easier and more likely that one or two bits will be flipped and based on this assumption the Higher the Hamming distance the better the CRC polynomial for that particular application [7]. The other way a CRC polynomial can differ from one another is the size.

Let's first explore CRC polynomial of degree 16. Baicheva et al. [8] try to compare different existing 16-bit CRC polynomial and identify the ones with the best performance in terms of detecting errors. The advantage to these 16-bit CRC code polynomials is that they are a lot smaller and therefore easier and quicker to calculate. Each different CRC polynomial is given a probability metric called the probability of undetected channel errors denoted by P_{ud} [8]. This P_{ud} is used to evaluate the effectiveness of the different 16-bit CRC codes. After performing the test on the selected 16-bit CRC codes, it was found that shortening the codes does affect the error detection performance [8]. In other words CRC codes that had a low order degree had poor P_{ud}. Switching over to CRC polynomial of degree 32 explored in the work of Philip Koopman [9] where he tries to find an ideal 32-bit CRC polynomial that will be effective for any message payload. He tries to fill the gap in the academic community by evaluating available CRC polynomials for different message payloads and their resulting Hamming distance (HD). He notes that if one were to test the effectiveness of a CRC code each time for a particular message payload it would

be too computationally expensive and so he tries to classify beforehand the ones that may best be suited for a particular situation [9]. After performing exhaustive exploration Koopman [9] present his results, which classifies each well-known CRC polynomial in a table with corresponding best match message payload size and the corresponding HD result. His result show that that the best error detection CRC code was a 32-bit CRC polynomial which he coined as the Koopman polynomial. From this work it is clear to see why 32-bit CRC polynomials are used as a standards for many applications. This also alludes to future work that could explore the idea of finding CRC polynomials that would be better suited for intentionally corrupting frames for the purpose of side channel communication. Going one step further, it may suggest that using a poor error detecting CRC polynomial may help hide the presence of side channel communication.

2.3 Related Works – Other Types of Side Channels

Steganography is a Greek word meaning covered writing [2]. This is different from encryption because steganography deals with hiding the secret messages from untrained eyes and encryption deals with masking or converting the secret message in a way that it is unreadable. Steganography or information hiding dates back to 440 B.C. where the Greeks used to tattoo secret messages on the messengers body with wax [2]. The main goal is to hide the presence of communication between the two parties. This is a one of the key characteristics of the side channel presented in this thesis.

Another great example of information hiding comes from the British. They were able to hide dots and microdots strategically placed in newspaper publications [2]. This

type of distribution of the secret message provided almost zero percentage for detection because to the normal reader would attribute these ink spots to the printing press markings [2]. Since then the field of Steganography has advanced and the techniques have been sculpted into many forms such as text, images, audio, video and importantly protocols. The key is to be able to exploit the characteristics and vulnerabilities of these forms in order to hide information [2]. This type of protocol steganography can also be referred to as a side channel. The common factor between all types of network infrastructures is that they are based upon the Open Systems Interconnection (OSI) model concept, which is made up of protocol layers communicating in a hierarchical stack. This hints to the notion that exploitation of one or more of these protocol stacks can create and establish side channel communication in existing network infrastructure.

Handel et al. [10] states that these side channels already exist in existing networks and warn that these channels can be inadvertently over looked and that administrators should be aware of them. In order to get a better understanding of possible types of side channels within the OSI model, it is advantages to look into a few, mainly the first two layers which relate closely the scope of this thesis work. Starting off with the first layer which is identified as the physical layer, two possible side channels can be created [10]. Due to the fact that this layer is the lowest level and deals with physical electrical signal processing devices, the side channel comes from exploiting these signal patterns. Specifically hiding the secret packet transmission by disguising it as legitimate signals or signal patterns [10]. The first type of side channel in the Physical Layer exploits normal signal patterns exhibit by different transmission protocols. For example during the

transmission control protocol (TCP) handshake, the signal should exhibit a signal pattern signature that contains fixed length signals in a specific sequence [10]. These patterns can then be exploited and imitated for use in to disguise secret data transmission to look like legitimate protocol signals.

Another type of side channel in the physical layer exploits the carrier sense multiple access with collision detection protocol (CSMA/CD) [10]. In this scenario the side channel is created by using other legitimate signals as a "piggy back" to carry the secret data. The idea is that by using the CSMA/CD protocol, any adversary nodes attached to this physical layer are able to identify when a collision has occurred because a "back off" signal has been issued characterized by the protocol [10]. This can be exploited to identify when a normal node is intending to transmit. When this time is determined, an adversary node can "piggy back" the transmitted signal by timing its own signal transmission just after or before the legitimate signal [10].

Now going up to layer two called the Data Link Layer, where unused frame space in the headers are exploited. Secret data can be hidden in these spaces and adversary nodes can modify their header checking algorithms to identify modified headers in those specific locations for intentionally hidden information [10]. Within this layer also exist error-handling protocol that use header space that could also be exploited. Both Mobile Ad Hoc Networks (MANETs) and Local Area Networks (LANs) are based on the OSI network model and thus contain vulnerabilities that provide possibilities for side channel communication. Note that all of these side channel examples are unique in their own way

but at the same time have the same underlying purpose of hiding the presence of secret data as opposed to encryption where the focus is to modify the data so that it's unreadable to the untrained eye. This is where each of these side channels are similar to the side channel presented here in this thesis.

Earlier works by K. Szczypiorski [11] have shown that side channel communication is possible by the exploitation of existing protocols within wireless and Ethernet environments. He proposed a way of creating a network steganography channel method that uses the vulnerabilities of the infrastructure. This is apart from the usual already seen Steganography methods such as hiding information in pictures, audio or video. Szczypiorski [11] terms this type of hidden channel or system as a Hidden Communication system for Corrupted Networks (HICCUPS). The idea behind this is to have a group of nodes that can communicate with each other using a predetermine way of intentionally corrupting a frames FCS fields. HICCUPS use the fact that any corrupted packets received are automatically dropped without question for efficiency purposes [11]. These groups of node will have to agree on a method of corrupting the frame in order to recover the frame for use. The author suggests using keys to encrypt the FCS to disguise it as being in error when the unknowing receiver node calculates the checksum. The other nodes outside the group not aware of the predetermined method of intentionally corrupting the frame will see that the frame is in error and will attribute it as noise. This idea is very much the bases of the side channel presented in this thesis. However this thesis explores a different method of intentionally corrupting the FCS field

using a predetermined CRC polynomial function. This methods avoids the difficult tasks of key exchange for use in the encryption algorithm which is a research topic on its own.

As mentioned earlier, the noise prone MANET environment would prove to be the best facility for this type of channel to exist but it can also be seen in a wired Ethernet Local area network (LAN) infrastructure. Jankowski et al. [12] proposed a way of creating a side channel within an Ethernet LAN infrastructure using improper Ethernet frame padding, address resolution protocol (ARP) and transmission control protocol (TCP) 3-way handshake. The LAN infrastructure does prove to offer a feature that MANETs do not have, which allows adversary nodes to identify other nodes that are fellow adversaries. This is done through the broadcasting ARP packets with improper frame padding. The padding bits is comprise of a random number not equaling to zero and the hash value of the medium access control (MAC) address based on the random number as the key. If there are other adversaries in the group they will broadcast their ARP packet with their random number and hash value in the padding bits. In order to perform the secret data transfer, the TCP segment packets are used with improper frame padding. The node that wants to initiate side channel communication will send a normal TCP segment request without improper frame padding directly to the other adversary node [12]. That adversary node will send an acknowledgement (ACK) back but this time it will improperly pad the frame with the secret message. The communication continues with exchange of ACK improperly padded segment packets. The sequences of TCP packet are exactly the same as normal 3-way handshake and to the outside observer this is seen as normal traffic. The author named this type of side channel communication “PadSteg”

[12]. This work helps identify that side channel does not limit itself to only wireless but may also exist in wired environments. This may help present possible future work that could take the side channel concept presented in this thesis and apply it in a wired environment. The natural question that arises is, if it is more easily detected due the centralized characteristics of the wired environment.

2.4 Related Works – Anomaly Detection Systems

The issue with having this unique side channel problem is lack of literature that is directly related to the detection. This section allows the reader to gain some insight into other types of side channels or attacks on a network and their countermeasures used to detect them. It acts as more of a general overview of the techniques and metric available for possible adaptation or implementation for detection specific to this thesis.

Most of the literature surveyed looked at intrusion detection systems that guard against Ethernet LAN based attacks. Zhang et al. [13] identify that there must be a distinction between wired and wireless based networks when it comes to the techniques used in intrusion detection. In other words, the techniques tried tested and true in a wired network cannot be ported over to MANETs. The main challenges that cause this un-portability stem from the decentralized and dynamic topology infrastructure of a MANET as mentioned before. In order for intrusion detection to work, the system must be able to find a normalcy state of network and be able to pin point an anomaly has been detected [13]. In order to discover a normal network state, the network data of the whole

network must be collected and analyzed [13]. In order for the intrusion detection system to be effective this network data must be collected in real-time with minimal delay.

Zhang et al. [13] identified that due to the decentralized infrastructure of MANETs this real-time traffic collection is not possible. This can only be done if the infrastructure is able to funnel or process data through a single gateway or end point [13]. Wired networks have this advantage where routers, gateways, firewalls and switches can become a centralized real-time data collector [13]. Instead the real-time traffic in MANETs is limited to collection by individual nodes that are within range of the communication traffic. This limits the capabilities of existing intrusion detection systems for use in MANETs [13]. In order to circumvent this data collection limitation, Zhang et al. [13] propose that there must be a cooperative detection system in place with groups of trusted nodes. These nodes must participate in intrusion detection notification by running local node-based algorithms [13]. The main feature of the algorithm is being able to calculate the level of confidence that there is an intrusion occurring based on its network data collected within range of its radio [13]. This shed some light into the challenges of working with MANETs versus Ethernet based LANs. It also identifies that existing intrusion detection techniques are more catered to a cooperative model which is found mostly in the Ethernet based LANS environment. This leaves some room for the discovery of new techniques and metrics that can be used in the MANET environment for detection of side channel communication.

Another form of anomaly detection is in the work of Hayajneh et al. [14] where they look into detecting the presence of malicious behaviours such as packet dropping and energy draining by nodes in wireless ad hoc networks. This malicious behaviour is plausible due to the combination of greedy nodes trying to limit power consumption and the lack of a central control center in wireless ad hoc networks [14]. Each node within the wireless ad hoc network relies on its neighbouring nodes to participate fairly in the forwarding of packets to reach their destinations. The threat model is based on the exploitation of the cooperative properties of nodes within the wireless ad hoc network infrastructure in order to perform two types of malicious behaviours [14]. The first of the two is malicious packet dropping (MD) after acknowledging receipt of the packet from the sender but does not forward the packet [14]. The motive to become uncooperative in forwarding these packets could its greedy nature to conserve power. The second type is similar to the first but without the presence of receipt acknowledgement, causing the sender nodes to retransmit, which may cause unnecessary energy drainage denoted by ED [14]. This type of behaviour does not have a meaningful purpose, but only to interfere with normal operations of nodes within the network [14].

Hayajneh et al. [14] then presented a five-step process that make up the detection protocol that the sending node can follow to determine possible malicious behaviour amongst its neighbours. The earlier step of the protocol tries to achieve a baseline sense of the network conditions through determining the probability of channel errors using pre-calculated signal to noise ratio (SNR) and channel errors based on counting the number of request to send (RTS) to clear to send (CTS) [14]. The assumption that the

authors had made is that if small RTS and CTS control packets that were sent within a window are corrupted, then the larger data packets sent within this window will also be corrupted with a high probability [14]. If acknowledgements are not received, the sending node can determine the likelihood that the receiving node is malicious and behaving in an ED manner by using the two network condition probability metrics stated above, even if indications suggest that packet corruption is low. As an aside this assumption is confirmed in the work of Xu et al. [15] where they try to find the effectiveness of using RTS and CTS handshakes in MANETs. They suggest that the existing RTS and CTS handshake that was put in place to mitigate medium sharing issues is not as effective as it was thought. They identify that the power needed to send out an RTS is lower than the power of successfully transmitting a packet. Xu et al. [15] also point out that the RTS power is enough to disrupt a transmitting packet. If there is a hidden node in range of the receiver and not in range of the sender, the RTS request sent by this hidden node will cause a collision with the packet. Xu et al. [15] suggest that instead of using RTS and CTS packets to limit the number of collisions, it more beneficial to measure the signal power of the RTS request. This is a better indicator as this measure power could be used to decide if it is enough power based on a threshold to successfully receive a packet. This confirms the work of Hayajneh et al. [14] as it identifies that the likelihood of a packet being corrupted naturally in transmission is low if the number of RTS and CTS are received without error as it takes less power to transmit them as Xu et al. [15] showed. Now to continue with detection of MD behaviour, the sending node must employ the help from surrounding nodes along with the two network conditions measures [14]. From the point of view of the sending node, it may have a false sense that its packets are

received without error and assumes that they are being forwarded when it receives acknowledgments. In actuality, the malicious node receiving the packets continues to send acknowledgements of receiving such packets, but do not forward them. In order to detect this type of MD behaviour, the sending nodes must depend on other nodes to watch for the forwarding of the packets that were acknowledged and accounted for [14]. The probability of both of these malicious behaviours, ED and MD, is calculated to a value the authors called P_m. If this P_m value is calculated and goes beyond a threshold, then the alarm is triggered. The concern here stems from how sensitive the detection protocol is based on the probability in packet corruption and channel errors. This suggests that if the wireless ad hoc network contains a very large amount of nodes, this will create more interference and noise, making detection very hard as the intentional packet dropping by the malicious node may be mistaken as normal due to the network conditions. This advocates that this detection protocol is only good for low interference, low noise wireless ad hoc networks. This is very similar to the problem that is posed here in this thesis because legitimate error can be used as a cover for side channel communication. It is now beneficial to briefly explore some prior works that addresses detection of the side channel that is related to this thesis.

Unlike intrusion detection, anomaly detection cannot base its features or metrics on abnormal usage behaviour patterns or profiles [16]. In the attempt of creating an intrusion detection system model, Denning [16] has listed some types of intrusions that could be alerted by abnormal usage patterns or profile. These include attempted break-in, masquerading/successful break-in, penetration by legitimate user, leakage by

legitimate user, inference by legitimate user, Trojan horse, virus, and denial-of-service [16]. In all these intrusion examples usage pattern can be compared to normal usage or legitimate users or processes [16]. For example, denial-of-service can be detection by very high activity or access to resources by a particular user when all other activity by other users are low [16]. Detection of the side channel anomaly does not have the same luxury of having a behavioural norm to compare with. As described, these side channel frames can hide itself amongst legitimate corrupted frames and becomes difficult to distinguish when the MANET environment is unpredictable and normally is prone to errors. This confirms that existing intrusion detection features or metric that exist cannot be directly transferred in the use for detection of the side channel and shows the need for new features and metrics to be discovered.

Li et al. [17] proposed the use of the frame error rate (FER) as a metric for detection of side channel. The idea was to measure the FER within a time window and if there is an abnormal increase in errors within that window then this may signal the presence of side channel [17]. The only issue is that the error still cannot be confidently identified as a side channel frames or legitimate errors cause by some change in the environment. Work done by Madtha et al. [18] proposed measuring Request to Send (RTS) and Clear to Send (CTS) control frames as a way of detecting side channel communication. The assumption made was that before every data frame was sent a RTS and CTS combination must be sent and received in order to avoid collision [18]. That means that for every data frame sent there will be corresponding RTS/CTS frame [18]. If there was no error in transmission the number of data frames will match the number of RTS/CTS frames

counted [18]. Conversely, if there was error in the transmission the number of data frames will be less than the number of RTS/CTS frames counted because the data frames would be seen as error [18]. The hypothesis that abnormally high RTS frames with low data frames will signal the presence of side channel communication because the side channel frames would be seen as error [18]. Again this metric is sensitive to real noise that can occur which could cause a high increase in the number of RTS frames compared to the data frames. Even though both of these prior metrics are sensitive to error that can occur naturally in an error prone environment, they are still very useful and can be used in combination with other metrics such as the Hamming distance metric proposed in this thesis. It is also important to understand the advantages and disadvantages of existing metrics in the attempt to emphasize the contribution of this thesis.

Chapter 3

Hamming Distance as a Detection Metric

One of the challenges when trying to come up with an anomaly detection algorithm or system for any domain specific problem is finding features or metrics that can be used that uniquely identify an anomaly. As it was seen in Chapter two, some metrics have been proposed in prior works that also explored detection of this specific side channel but they proved to be sensitive to certain factors that exist in normal wireless environments. This Chapter starts off with some insight into the motivations that led to the discovery of the Hamming distance as a possible metric for use in the detection and finally, include an explanation on how it could be used.

3.1 The Delta Cyclic Redundancy Check Metric

The direction towards the discovery of the Hamming distance as a metric can be accredited to an unpublished survey written by a colleague Dr. John K. Jacoub which compared existing anomaly detection algorithms and techniques. At the end of the survey he provided some very early preliminary work that proposed a technique which was investigated further in this thesis. The technique proposed was based on the hypothesis that each CRC polynomial when used in the CRC generator function would map to its own set of CRC values and if the numerical difference between two values

within the set were taken then it would yield a significantly different result than the difference between two values out of the set. He termed this difference value as the delta CRC (ΔCRC). If these ΔCRC values are plotted and the CRC values come from different CRC polynomial functions then there should be two distinct clusters. Let's redefine the hypothesis in a side channel scenario:

Let's first define the following for purpose of reference:

- *There exist two CRC polynomials called G and H*
- *There also exist some frame data defined in the form:*
  - $a_n x^n + \ldots + a_2 x^2 + a_1 x^1 + a_0 x^0$ *, where* $a_k \in \{0,1\} \; \forall \; k \in \mathbb{Z}, 1 \leq k \leq n$ *and,*
  - *n is the highest degree of the polynomial*
- *And,*
  - $CRCG = (a_n x^n + \ldots + a_2 x^2 + a_1 x^1 + a_0 x^0) \bmod G \mid CRCG \in A$ *, where* $A \subset \mathbb{Z}$
  - $CRCH = (a_n x^n + \ldots + a_2 x^2 + a_1 x^1 + a_0 x^0) \bmod H \mid CRCH \in B$ *, where* $B \subset \mathbb{Z}$
- *And A and B are disjointed sets such that* $A \cap B = \emptyset$*, where* $\emptyset$ *is the empty set*

Then based on the proposed hypothesis, if polynomial *G* is used by all normal nodes in a MANET then the adversary node uses polynomial *H* to purposely corrupt the frames in order to create the side channel. All unsuspecting normal nodes will see all frames that are coming from the adversary node as corrupted because the calculated *CRCG* will not match the received *CRCH*. However if the nodes were smarter and could perform ΔCRC calculation then the following possible outcomes could arise.

List of possible outcomes:

1. The frame was determined to be without error, then calculated *CRCG* value and the *CRCG* value received in the frame would match and the ΔCRC result would be 0.
2. The frame was determined to be with error, meaning the calculated and the received CRC in the frame did not match, then there are two possible outcomes.
    2a. The frame data was naturally corrupted, then the calculated *CRCG* value minus the *CRCG* value received in the frame will not be 0 but yield a ΔCRC closely resembling elements in the set *A*.
    2b. The frame data is fine but the FCS field is purposely corrupted and contains the *CRCH* value which was calculated by some unknown polynomial *H*, then the calculated *CRCG* value minus the received *CRCH* value in the frame will not be 0 but yields a ΔCRC that are significantly different then elements of both *A* or *B*.

If both outcomes 2a and 2b were plotted this would yield a cluster of elements that closely resembles elements in set A and elements that are significantly different then elements of both *A* or *B* which would indicate the presence of a side channel. More specifically, the existence of adversary nodes using a different unknown CRC polynomial *H* to purposely corrupt its frame before transmission. If the above statements in the hypothesis can be proven to be true then the ΔCRC can be used as a very effective metric for detection of side channel. In order to test this hypothesis a simulation was created in

MATLAB/Simulink [19] that reproduced the above scenario using real life CRC32 polynomials.

Let's first entertain that the hypothesis is true, then running a simulation that produces the outcomes describe above will yield two distinct clusters which in a histogram will produce two different peeks. MATLAB/Simulink was used to test this hypothesis and the simulation can be seen in Figure 3.1 below. Note that the configuration for each module in the simulation below can be found in Appendix A for further reference.

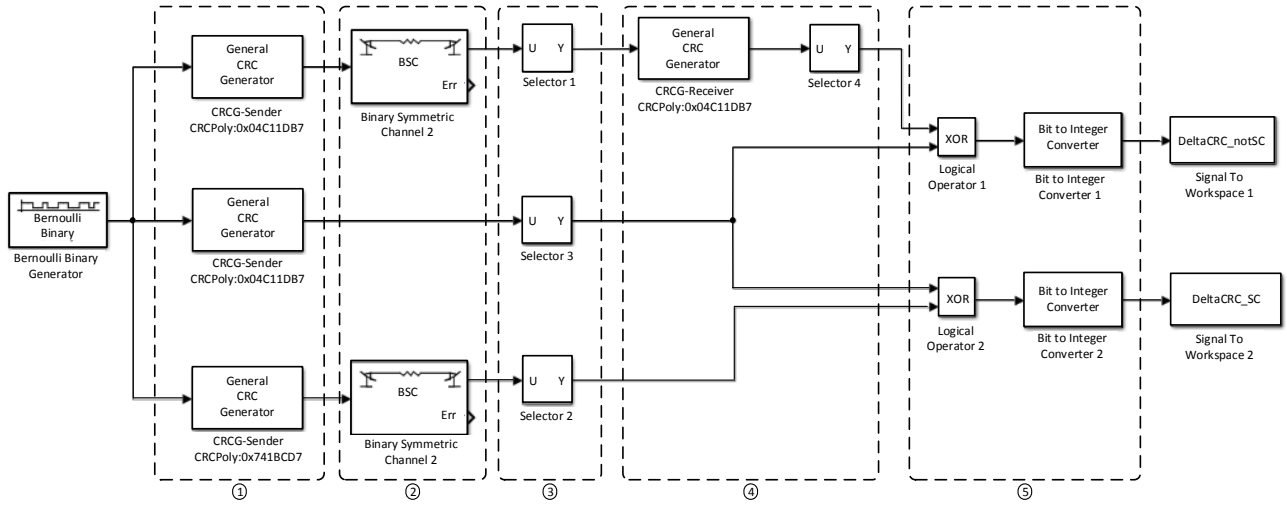


Figure 3.1 MATLAB/Simulink Simulation for Testing Delta CRC Hypothesis

The simulation starts from the far left where the Bernoulli Binary generates 100000 frames of size 1024 bits. The number of frames can be specified by modifying the simulation stop time at the top of the model window. One frame will be generated per

one second simulation time and is copied three times for input into three CRC generator paths in section 1. Generator labelled "CRCG-Sender" uses the default CRC32 polynomial 0x04C11DB7 to generator a CRC value and tags it on to the end of the frame. The frames that go through this generator will simulate a frame that will be transmitted. Generator "CRCH-Koopman" uses another CRC32 polynomial 0x741B8CD7 and any frames that go through this generator will simulate a transmitted side channel frame that is purposely corrupted. Generator "CRCG-Verify" is used as a verifier and also uses the default CRC32 polynomial. The CRC value generated will be used at the end of the simulation to compare with the other generated CRC values from the other two frame paths. Section 2 has two Binary Symmetric Channels which simulate frames being transmitted with probability of being erroneous. Channel 1 handles all the non-side channel frames transmitted while channel 2 handles all the side channel frames. Remember that the frames that go into these channels are the same frames but have different CRC values. Section 3 has three selectors that are used to pick out desired bits from the whole frame. Selector 2 and 3 both pick out the CRC bits that were tagged to the end of the frame and send it off for ΔCRC calculation in section 5 of the simulation. Selector 1 has a different task, it will pick out the data portion of the frame (1024 bits) and send it to another CRC generator module labelled "CRCG-Receiver" in section 4. This section 4 simulates the receiving nodes task of recalculate the CRC value using the same CRC32 default in order to see if the frame has been corrupted naturally through transmission. The selector 4 in this section 4 will now pick out the CRC value and also send it off for ΔCRC calculation in section 5. The ΔCRC calculation is handled by both the XOR logical operators which will perform binary subtraction on the CRC values inputted.

Logical operator 1 calculates the ΔCRC values for the non-side channel frames performing the binary subtraction on CRC values that were calculated using the default CRC. Referring back to the outcomes listed above, the ΔCRC outcomes would be 0 for frame with no error or not 0 if the frame was naturally corrupted. The ΔCRC outcome described in bullet 2b from above is calculated in logical operator 2 when the CRC value generated form the default CRC polynomial is binary subtracted from the CRC values generated from the koopman CRC polynomial. All the resulting ΔCRCs are outputted to the workspace "DeltaCRC_notSC" and "DeltaCRC_SC" for plotting. Remember that if the hypothesis holds true then there will a distinct peaks for "DeltaCRC_notSC" and one for "DeltaCRC_SC". The resulting histogram of the CRC values plotted for each generated frame can be seen below in Figure 3.2.

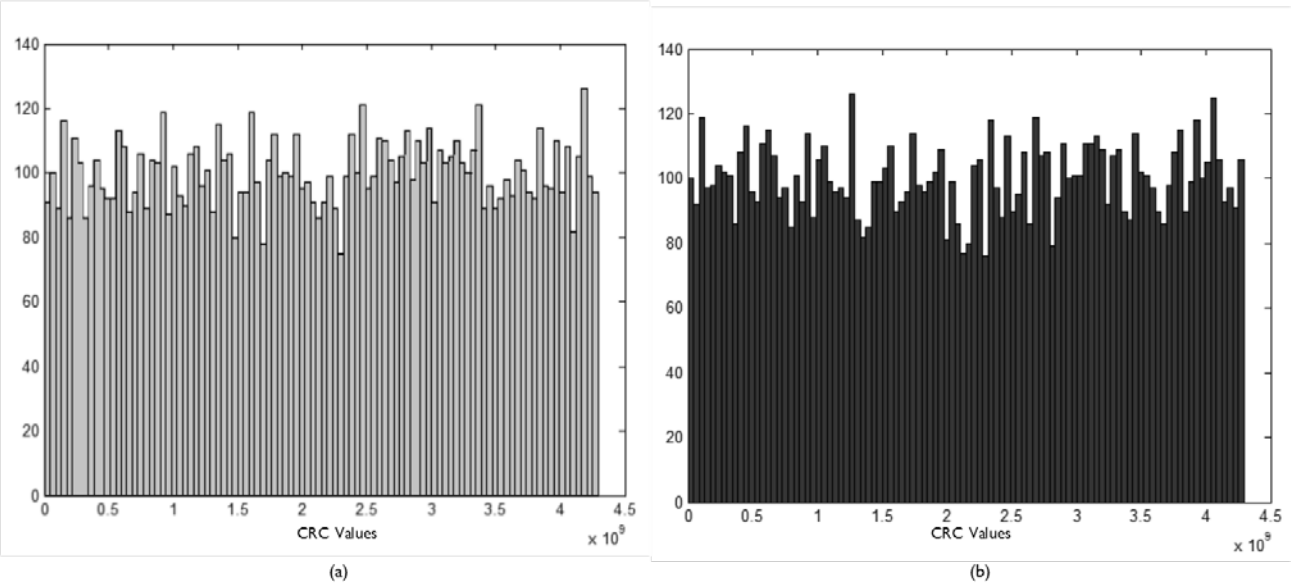


Figure 3.2 Delta CRC Hypothesis Testing Results. (a) CRC values generated by the Default polynomial and (b) CRC values generated by the Koopman polynomial

Figure 3.2 shows histogram of both ΔCRC for both non-side channel (left) and side channel frames (right) on separate graphs. If the hypothesis was true, there should be two clear peaks seen. What is actually seen is that the CRC values seem to be distributed evenly for both polynomials. Note that the results here are from using CRC32 Default and Koopman and cannot be generalized to other unknown CRC polynomials. Unfortunately, these CRC32 polynomials are standards that are known to be used in real implementation and the hypothesis have to hold true for at least these CRC polynomials. The same scenario was ran but this time substituting the Koopman CRC polynomial for Castagnoli and it yielded the same results. Further work could be done to find CRC polynomials that could possess the desired characteristics describe above to prove the hypothesis to be true but this is outside the scope of this thesis and is left as possible future work.

3.2 The Hamming Distance Metric

Taking the same direction but instead of finding the numerical difference of two different CRC values the idea was to take the Hamming distance instead. The Hamming distance is a common metric used in code theory to compare two different bit strings [20]. The Hamming distance measures the number bits that are different between the two bit strings [20]. What is interesting about the Hamming distance is that it is commonly used for error correction [20]. A method called the "nearest neighbor decoding" uses the Hamming distance to figure out what possible code word was sent if the received code word was in error [20]. This method states that the code word that was sent but was in error can be identified by the smallest Hamming distance value between the error code

work and all its possible code words [20]. This is under the assumption that the channel is not that noisy [20]. This assumption is a valid one to make as there would be no point to communicate if the channel is so noisy mostly all the frames are lost. Even for an adversary a noisy channel is not ideal when trying to move data around. This assumption that the number of bits flip are minimal during transmission paired with the knowledge that CRC polynomials are hash functions designed to avoid as many collisions as possible, sparked the idea of using Hamming distance as a possible metric for detection of side channel frames. A proposed way of detecting these intentionally corrupted side channel frames would still be the same as before which was looking into the CRC values and comparing the difference between the received value and the calculated value but instead measure the Hamming distance (HD) between the two CRC bit strings. The new hypothesis to test would be that the HD value of a naturally corrupted is significantly low compared to an intentionally corrupted frame. This is under the following assumptions:

1. The CRC polynomial used is one that is known and used in real implementation such as the Default, Koopman or Castagnoli describe earlier.
2. The adversary will not use encryption or change the integrity of the frame data and the only portion that is different is the FCS containing the CRC value generated by a different CRC polynomial.

The MATLAB/Simulink model, shown in Figure 3.3 was modified to test using the Hamming distance metric instead of the ΔCRC. It was also modified to include the

Castagnoli CRC32 polynomial as well as changing the output to the workspace as a CRC bit string instead of the numerical values for use in the HD comparisons. Note that the configuration settings of each of the module is the same as the model from Figure 3.1 and can be references in the same Appendix A.

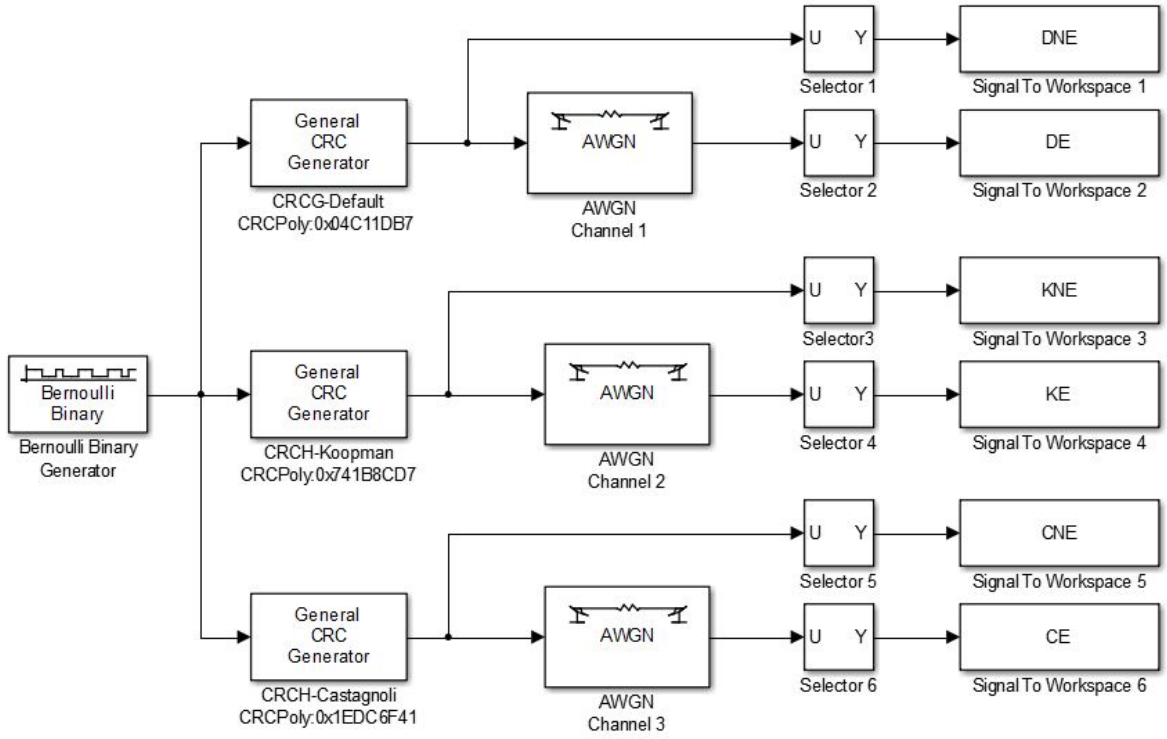


Figure 3.3 MATLAB/Simulink Simulation for Testing Hamming Distance Metric Hypothesis

The purpose of the simulation model in Figure 3.3 is to test two things. One is to confirm that during transmission the number of flip bits throughout the frame is going to be minimal. Assuming that the channel should not be too noisy or no nodes will be able to communicate properly including the adversary nodes. The other is to confirm the

variability of the CRC values generated by the three different CRC polynomials which is the consequence of the hashing function characteristics that was mentioned earlier. Instead of verifying the variability of the numerical values of each CRC value produced, the Hamming distance variability will be measured between the values generated by the different CRC polynomials. In other words, measure the number of bit differences between the bit strings produced by the default CRC versus Koopman and Castagnoli. If each bit string produced by each CRC polynomial is very different to avoid collisions then the HD values between each of them will be very high. Higher with respect to the HD value obtained between a naturally corrupted CRC bit string versus an uncorrupted bit string generated by the same CRC polynomial. If both these test are confirmed true then there is a good chance that the hypothesis is true that the HD value of a naturally corrupted frame is significantly lower compared to an intentionally corrupted frame using a different CRC polynomial. Now let's define some of the variables that are outputted to the workspaces:

a. DNE (Default No Error): *CRC bit generated by default CRC32 polynomial.*
b. DE (Default Error): *CRC bit generated by default CRC32 polynomial but in error.*
c. KNE (Koopman No Error): *CRC bit generated by Koopman CRC32 polynomial.*
d. KE (Koopman Error): *CRC bit generated by Koopman CRC32 polynomial but in error.*
e. CNE (Castagnoli No Error): *CRC bit generated by Castagnoli CRC32 polynomial.*
f. CE (Castagnoli Error): *CRC bit generated by Castagnoli CRC32 polynomial but in error.*

With those variables defined, the following HD comparisons were made:

1. DNE vs DE
2. DNE vs KNE
3. DNE vs KE
4. DNE vs CNE
5. DNE vs CE

The first comparison yields the HD values when the possibility of natural corruption is added to a frame where the default CRC32 polynomial was used to generate its CRC bit string. Consequently, because the CRC bits strings are calculated on the same frame data and using the same polynomial then the frames that are not in error will have a HD value of zero. Conversely, the only case where the HD is greater than zero is when the frame was naturally corrupted. Comparisons 2 and 3 produces HD values between CRC bit strings generated by the default polynomial versus CRC bit strings generated by Koopman polynomial. The difference is comparison 3 adds the possibility of natural corruption. The last two comparisons 4 and 5 are identical to 2 and 3 but instead use the Castagnoli polynomial. Intuitively, the HD value produced for these last four cases should be greater than zero because the use of the a different CRC should guarantee that at least one bit in the CRC bit strings is different otherwise they would be the same. In order to test this the model in Figure 3.3 is run with two variables. The first one is the size of the frame and the second is the number of frames generated. The size was varied from 512 bits to 2048 bits with increments of 512 bits (512, 1024, 1536, 20148) to see if the HD was affected by the size of the frame. The number of frames used was fixed to 1,000,000.

In order efficiently process a large amount of frames, a MATLAB script was developed that calculated the statistical measures such as the mean, median, mode, variance, standard deviation and coefficient of variation. This MATLAB script can be found in the Appendix A for further reference. Mostly all of these statistical measure are familiar except for coefficient of variation which might need to be defined. The coefficient of variation is a ratio of standard deviation to mean ($\frac{standard\ deviation}{mean}$). It gives context to infer meaning out of the resulting standard deviation. In other words it answers the question of how well the resulting standard deviation represents the spread or variability of the data with respect to the mean. The closer the coefficient of variation is to 1 then the greater the spread or variability of the data. The statistical result of the four simulation runs with different frame sizes can be found in the Tables 3.1, 3.2, 3.3, and 3.4 below.

Table 3.1 Statistical Results for 512bits Frame Size

Frame size: 512 bits	DNE vs DE	DNE vs KNE	DNE vs KE	DNE vs CNE	DNE vs CE
mean	8.034	15.997	15.996	16.003	15.999
median	8.000	16.000	16.000	16.000	16.000
mode	8.000	16.000	16.000	16.000	16.000
variance	6.021	8.011	7.984	7.993	8.003
standard deviation	2.454	2.830	2.826	2.827	2.829
coefficient of Variation	0.305	0.177	0.177	0.177	0.177

Table 3.2 Statistical Results for 1024bits Frame Size

Frame size: 1024 bits	DNE vs DE	DNE vs KNE	DNE vs KE	DNE vs CNE	DNE vs CE
mean	8.033	16.002	15.998	16.001	15.999
median	8.000	16.000	16.000	16.000	16.000
mode	8.000	16.000	16.000	16.000	16.000
variance	6.017	8.016	8.013	8.004	7.993
standard deviation	2.453	2.831	2.831	2.829	2.827
coefficient of Variation	0.305	0.177	0.177	0.177	0.177

Table 3.3 Statistical Results for 1536bits Frame Size

Frame size: 1536 bits	DNE vs DE	DNE vs KNE	DNE vs KE	DNE vs CNE	DNE vs CE
mean	8.040	15.997	16.001	16.006	16.006
median	8.000	16.000	16.000	16.000	16.000
mode	8.000	16.000	16.000	16.000	16.000
variance	6.022	8.009	8.002	8.018	7.990
standard deviation	2.454	2.830	2.829	2.832	2.828
coefficient of Variation	0.305	0.177	0.177	0.177	0.177

Table 3.4 Statistical Results for 2048bits Frame Size

Frame size: 2048 bits	DNE vs DE	DNE vs KNE	DNE vs KE	DNE vs CNE	DNE vs CE
mean	8.039	16.001	16.005	15.996	16.002
median	8.000	16.000	16.000	16.000	16.000
mode	8.000	16.000	16.000	16.000	16.000
variance	6.012	7.993	7.978	8.006	7.996
standard deviation	2.452	2.827	2.825	2.829	2.828
coefficient of Variation	0.305	0.177	0.176	0.177	0.177

Analyzing the results of the simulation it can be seen that there is a significant average HD value difference between the CRC bit strings generated by the default polynomial versus both the Koopman and Castagnoli polynomials. The consensus

between all four simulations is that the resulting HD seems to be averaging around 8 for DNEvsDE and all other comparisons 16. The most important thing to notice is that based on the low coefficient of variant values, the HD values are pretty uniformed. These results confirm with greater promise that the HD value of a naturally corrupted frame is significantly lower compared to an intentionally corrupted frame using a different CRC polynomial. Based on the experimental variables it is hard to confidently generalize the results to more real life scenarios. Even though these preliminary results look promising more work must still be done in validating and evaluating this Hamming distance metric.

Chapter 4

Hybrid Testing Environment

The most challenging part about setting up a testing environment for this domain specific research area is the lack of existing real life implementation or protocols that supports the delivery of intentionally corrupted frames to form a side channel. The obvious direction is finding existing hardware or software solutions that may be modified to support side channel communication. In order to understand the direction and motivation in developing a test environment, let's first explore some earlier works that surveyed suitable test environment to facilitate side channel communication. This is followed by some insight into preliminary work using the QualNet simulator which is intended to spark or supplement future work. Finally, an introduction and overview of the proposed hybrid testing environment is given for use in the validation and evaluation of the effectiveness the Hamming distance metric for detection.

4.1 Hardware versus Software as a Test Environment

In some earlier works, Najafizadeh et al. [21] discovered that it was not possible to implement side channel communication on the existing network cards that are based on wireless protocol standards 802.11. This was due to the fact that the FCS calculation found in the MAC Layer is hardwired into the communication chipset [21]. Making it impossible to change the CRC polynomial within the FCS calculation. However they do

reference an older discontinued network card model DWL-AG530 produced by D-Link that uses a chipset called the Atheros AR5212 which supports modifications to its MAC Layer [21]. This highlights the fact that it can be possible to custom build chipsets that do not follow standards and would allow for alterations of the CRC polynomial within the FCS calculation. Unfortunately due limited resources, implementing side channel communication in existing hardware that follow the 802.11 standards would be impossible for the purpose of testing in this thesis work. So the natural progression is to look into simulation as a software solution. The advantage to having a network simulator as a test environment is that it is scalable and it's cheaper to modify software components than hardware components. Another advantage is that most of these network simulators are able to provide communication channel models, routing protocols and mobility models. The issue is finding one that can be modified to add the capabilities of side channel communication. Most simulators are event/message driven meaning that the concept of time is simulated. This means that the actual transportation of packets between nodes are simulated. In other words the time is advanced based on the calculated time for each packet to travel between nodes taking into account factors such as distance and size of the message. Alluding to the fact that the actual packet is never really sent which means that error occurring on the packet is also simulated based on the channel model probabilities implemented. For example if a packet is deemed to be in error based on the channel model, the actual frames bits do not reflect the corrupted bits, which confirms that these simulators do not have any concept of an FCS field for error handling. The other requirement is to be able to get access to the frame contents in order to include an FCS field for error handling.

Some previous works have surveyed a number of simulators such as NS-2 [22], QualNet, Sinalgo and MATLAB/Simulink that will help as a starting point in finding an environment suitable to facilitate side channel communication. Odor et al. [3] looked into simulators such as NS-2, QualNet and MATLAB/Simulink. They found that both NS-2 and QualNet have the same stochastic model on handling frames that were found to be in error [3]. With some previous work in collaboration with DRDC, Odor et al. [3] did determine that they could not modify the frame error handling in NS-2 due to the difficulty of accessing the contents in the frame. However they did not mention any further work in QualNet to modify the frame error handling to facilitate the addition of side channel communication [3]. Odor et al. [3] also present some initial work with MATLAB/Simulink's communication toolbox where they were able to implement a test environment that included all the requirements needed for side channel communication. This included a way to corrupt bits within the frames found to be in error by using the Rican Fading channel model [3]. This thesis explores further into both QualNet and MATLAB/Simulink as potential simulators. Najafizadeh et al. [21] used Sinalgo as their base simulator for their testing environment. The reasoning for choosing Sinalgo was that it was very well document and easy to modify [21]. Najafizadeh et al. [21] were able to successfully add an FCS field and implement a FCS generator function however the downfall of Sinalgo is that it does not come with a full network stack implemented; in particular it is missing the 802.11 CSMA/CA [21]. This required them to implement a simple communication model to reduce collisions [21]. Instead of fully implementing the 802.11 CSMA/CA protocol, they opted to implement a communication strategy that

modelled after the poison distribution [21]. This served its purpose but still left room for finding another simulation that was more true to existing standards in real life communication which started further investigation into QualNet as a possible software solution.

4.2 QualNet Network Simulator

QualNet by Scalable Technologies [23] was very appealing as a potential candidate in providing a test environment because it offered characteristics and behaviours of a real life network. It includes capabilities that allow for any existing wireless hardware to be realized in the simulator by offering a large amount of customizable parameters and specifications within different available devices or nodes. It takes into account many different factors for its communication broadcasting model such as weather, terrain, distance, and radio strength. Many industry leaders in the communication business have used QualNet to test their own network infrastructure for scalability and reliability [23]. Through its scalability, large numbers of nodes can be added and simulated with results that closely mimic a real life network [23]. It also includes a mobility model where paths could be randomly generated or explicitly specified through way points. Along with this, it also supports all kinds of routing protocols including OLSR which was mentioned earlier as ideal for MANETs. One other favourable advantage of QualNet was that it offered a fully functional network protocol stack that mirrors the real life hardware protocol implementation which most other simulators did not [23]. The only downfall was that it did not provide any form of data output for offline processing and analysis. QualNet also only provided some minimal stats reporting that was printed to a file.

Despite this only downfall, as a whole QualNet appeared to allow for the most realistic modelling of the MANET environment to facilitate side channel communication. It is obvious that some modifications are still needed to be able simulate side channel communication for testing.

The initial modifications, in collaboration with DRDC, was comprised of adding data dump capabilities for offline processing and adding metrics such as dropped frames, good frames and frame error rates for a specified time window. This initial work was featured in the 2013 proceedings of the Military Modeling & Simulation Symposium called "A realistic implementation for simulating side channel in mobile ad hoc networks" [17]. The scope of that work was to investigate detection of side channel communication through measuring the frame error rate (FER) fluctuations [17]. This first implementation of side channel communication was done in the application layer focusing on the constant bit rate application (CBR) [17]. The CBR application code was modified so that for every second application link created from node to node, that link was marked as side channel communication [17]. The implementation of side channel communication in this way was in line with how QualNet handles frames in error which did not require the modification of the simulator's original stochastic frame error handler mentioned earlier [17]. This served its purpose at the time as the only interest was on the number of frames in error and not how many bits in the frames were corrupted [17]. With the new scope of this thesis, further modifications to QualNet were needed to allow for a more realistic way of handling error frames. This would include the addition of an FCS field within the frame, the implementation of an FCS generator function based on any

CRC polynomial as input, and the ability to flip selected bits of the frame if it is deemed erroneous. The implementation was planned to have two sequential phases. The first phase was to implement the FCS generator function which includes the addition of an FCS field within the frame. The second phase was implementation of the function to flip bits in the frame if it is deemed erroneous. The first phase was completed but when tested with a simple two node scenario having one communication link, it failed only running a quarter of the way. After a considerable amount of time spent debugging and testing, the root cause of the issue was still found. This prompted the decision to stop any further modification of QualNet and set the focus on MATLAB/Simulink for setting up a testing environment.

As seen in the previous chapter describing the preliminary work in discovering the Hamming distance as a possible metric for detection of side channel communication and also work by Odor et al. [3], MATLAB/Simulink is capable of facilitating side channel communication. It does so by providing a communication toolbox that can be used to fulfilling the requirements outlined earlier, which included the addition of an FCS field within the frame, the implementation of an FCS generator function based on any CRC polynomial as input, and the ability to flip selected bits of the frame if it is deemed erroneous. The disadvantage of using this communication toolbox is that it cannot simulate multi-node scenarios because there is no 802.11 CSMA/CA to handle multiple communication links. So the capabilities is limited to one node generating fixed sized frames using the “Bernoulli Binary Generator” which can be seen at the far left in Figure 3.2 in the previous chapter. As a consequence the idea of routing and mobility is not

available as only one sender and receiver node can technically be simulated. In turn this means that there is also no mobility model to allow for a MANET environment that was described earlier as being an ideal testing against side channel communication. For the preliminary work described in the previous chapter of discovering and evaluating the Hamming distance this implementation served its purpose. However in a real life scenario two things need to be realized, first is the type and size of frame will vary and second there are usually more than one node in a network. Another unrealistic implementation parameter not evident in the model in Figure 3.2 is the SNR value used by the "AWGN " channel was set before runtime and the same value is used thorough out the simulation. In a real life situation this value could vary based on the signal to noise sensed from the network at that particular moment in time. This is important to understand because the SNR value will affect the number of possible bits being flipped in the frame, which in turn will directly affect the Hamming distance value. In order to keep the simulation as true to real life as possible some modifications to the original simulation model shown in Figure 3.2 is needed.

4.3 The Hybrid Approach

In order to encompass all the real life elements into the MATLAB/Simulink, a new hybrid approach had to be taken. This hybrid approach enlists the help of the AirPcap Network adapter [24] and Wireshark [25] to capture network data and frames. This includes data from multiple nodes in the network. This becomes the real life part that was added to the simulation making up the whole testing environment. Before presenting the final testing environment shown in Figure 4.1 that follows, it will be beneficial to first give a quick

overview of Wireshark and the AirPcap Network adapter and why it was added to the testing environment. Wireshark is a well-known network protocol analyzer tool that is used by many I.T network administrators to monitor and troubleshoot issues in their network [25]. It has the capability of capturing and saving network states at a given period of time for later processing [25]. These network states are comprised of captured time stamped frames with detailed information such as header and protocol information [25]. In order to satisfy the second realistic element described earlier, the SNR value must also be captured from the network for use as input into the "AWGN" channel. This ensures that the probability of the number of bit flips is dependent on the real SNR value sensed from the network for each frame captured and not dependent on a fix SNR value. In order to obtain this value from the capture using Wireshark in the Windows environment, the AirPcap network adapter was used [26]. This AirPcap allows for full raw 802.11 captures in the Windows environment and the SNR value was extracted from each frame captured. AirPcap allows Wireshark to capture not only good frames but also extend the capability to capture erroneous frames as well. This is used to get an idea of the percentage of frames lost in transmission for a given capture. With these capabilities at hand, it seemed to be a natural addition into the testing environment in order to keep it as realistic as possible. The only aspect that is not realised in this hybrid testing environment is mobility. As it was mentioned earlier the MATLAB/Simulink simulator lacks a mobility model and therefore must be captured in the real portion. The challenging part is that a lot of parameters must be accounted for when dealing with mobile nodes such as movement patterns and speed that could affect the overall performance. This could yield confounding variables that could skew the results. The

other challenging thing about mobility in a real life scenario is being able to accurately define each parameter so that the experiment can be duplicated for later verification or testing. Due to the added complexity that mobility might add and the already unpredictable nature of wireless it was decided to keep the scenarios static. This will allow the thesis to fully focus on the evaluation of the Hamming distance but in a wireless ad hoc network instead. This offers the opportunity for future work to investigate the effectiveness of the Hamming distance metric in a MANET where the parameters can be observed more carefully. The only thing left now is to explain how the data from Wireshark is filtered and formatted for use in the MATLAB/Simulink simulation portion. This explanation will be included in the overall overview of the testing environment of Figure 4.1.

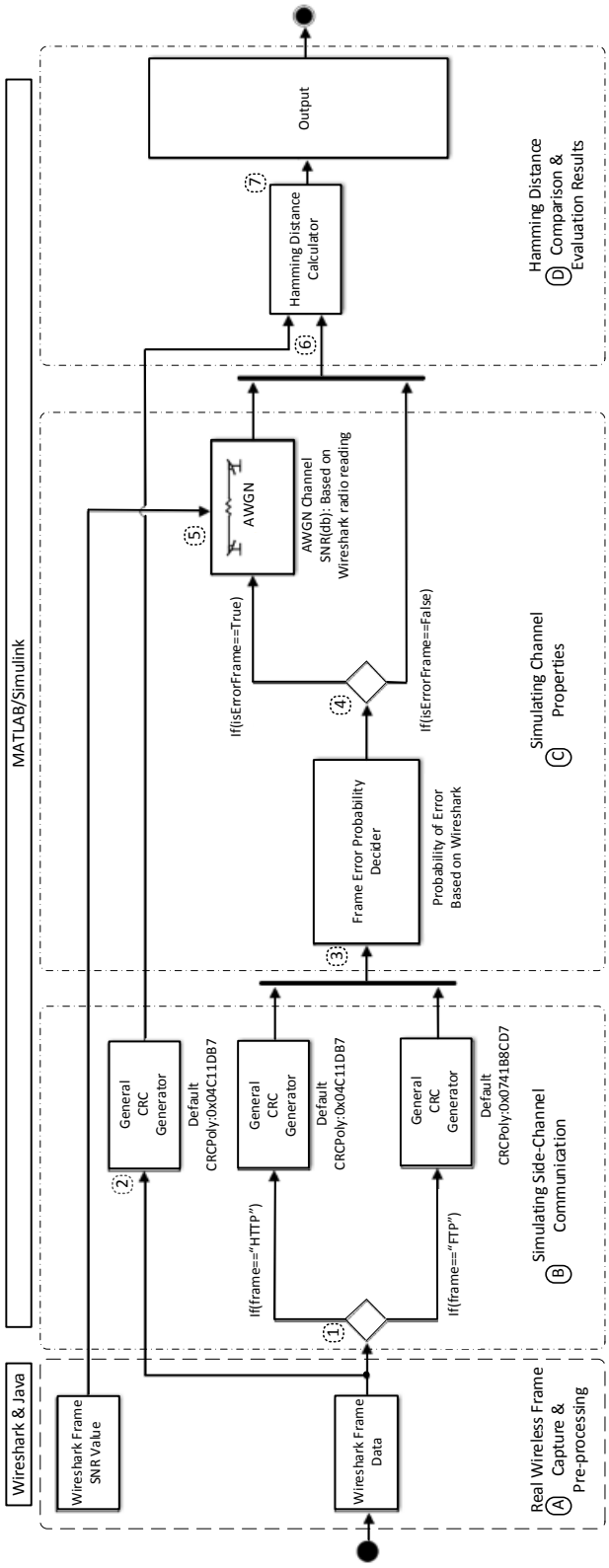


Figure 4.1 Hybrid Test Environment Model Flow Diagram

As it can be seen from Figure 4.1, the testing environment is comprised of the "Wireshark & Java" and the "MATLAB/Simulink" as the main sections. The MATLAB/Simulink section is split further into three subsections identified by letters B, C, and D in the same figure. The Wireshark & Java section is given a letter A identifier not to indicate a subsection consisting within but for the only purpose of ease of reference when referring to the figure. Starting off with section A from Figure 4.1, this section makes up the real portion of the testing environment. It is responsible for the capturing of real wireless frames using Wireshark and performs pre-processing using the custom Java application developed to format the data for input into the MATLAB/Simulink portion. Figure 4.2 found below illustrates the processes that are involved this first section A.

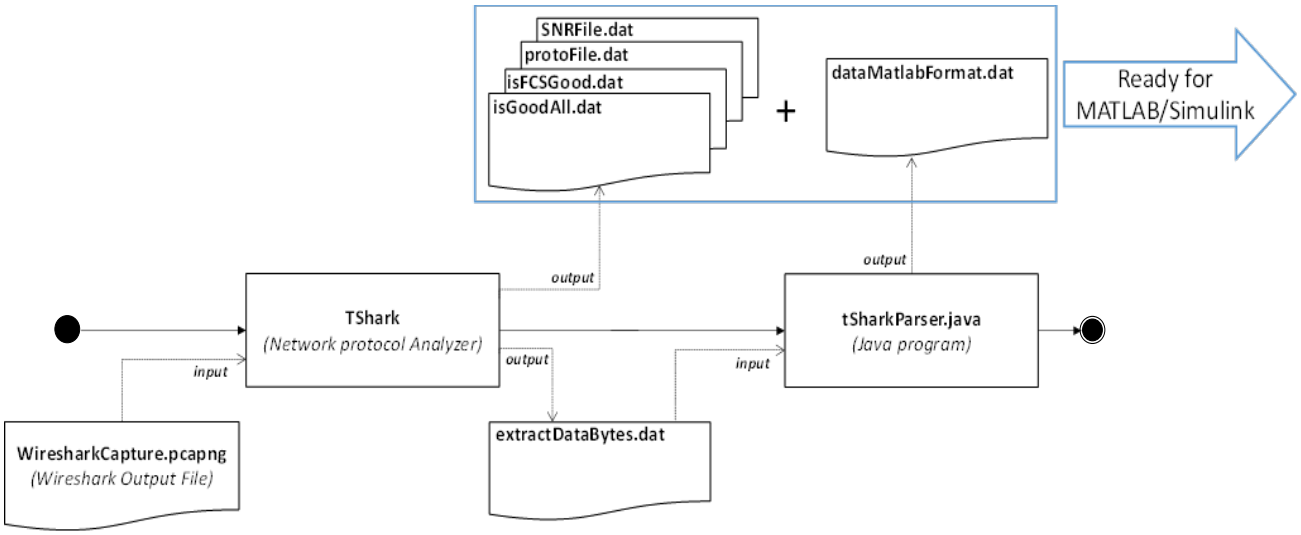


Figure 4.2 Wireshark & Java Process Flow Diagram

As it can be seen from Figure 4.2, there are two main processes. The first process uses the captured Wireshark inputs it into the TShark network protocol analyzer. This TShark network protocol analyzer is a just a command line version of Wireshark and only used for ease of exporting filtered data using the redirection operator available in Windows command prompt terminal. The TShark filtering commands used to produce the five ".dat" output files seen in Figure 4.2 are listed below with:

```
cmd1. tshark -r WiresharkCapture.pcapng -T fields -e wlan.fcs_good >> isGoodAll.dat

cmd2. tshark -R "(ip.src==<IPaddress> && http) || (ip.src==<IPaddress> && ftp)" -r
WiresharkCapture.pcapng -T fields -e wlan.fcs_good >> isFCSGood.dat

cmd3. tshark -R "(ip.src==<IPaddress> && http) || (ip.src==<IPaddress> && ftp)" -r
WiresharkCapture.pcapng -T fields -e tcp.srcport >> protoFile.dat

cmd4. tshark -R "(ip.src==<IPaddress> && http) || (ip.src==<IPaddress> && ftp)" -r
WiresharkCapture.pcapng -T fields -e radiotap.db_antsignal >> SNRFile.dat

cmd5. tshark -R "(ip.src==<IPaddress> && http) || (ip.src==<IPaddress> && ftp)" -r
WiresharkCapture.pcapng -x >> extractDataBytes.dat
```

Figure 4.3 TShark Filter Commands to Generate Data Files

The first TShark command "cmd1" is issued to redirect filtered results into the "isGoodAll.dat" which contains the status in binary format of each frame received with value 1 as good and 0 as erroneous. This file is used for calculating the percentage of frames in error for the Wireshark capture for the whole network. The second command "cmd2" is issued to redirect filtered results into the "isFCSGood.dat" file which contains the same binary status output describe for "cmd1" but it filters out only the frames sourced from the specified IP address. This is used to identify all the frames received without error to ensure that the frame is uncorrupted for manipulation later in the

MATLAB/Simulink portion. The handling of these frames will be explained later. The third command "cmd3" redirects filtered results into the "protoFile.dat" file which contains the port value 21 for FTP or 80 for HTTP protocol for each frame sourced from the specified IP address. The fourth command "cmd4" redirects filtered results into the "SNRFile.dat" file which contains the SNR value sensed for each corresponding frame captured sourced from the specified IP address. The last command "cmd5" redirects filtered results from TShark into the "extractedDataBytes.dat" file which contain the Wireshark Byte hex dump for each frame sourced from the specified IP address. As Figure 4.2 shows, this last file does not go directly into the simulation portion, instead it is used as input for the second process.

This second process is a java program that was custom built to take the Wireshark Byte hex dump of each frame and reformatted it into a serial bit representation. An example of the reformat of one frame can be seen below in Figure 4.4 where the Wireshark Byte hex dump frame is on the left and the resulting serial bits representation is on the right. The separation lines found in this figure were added to clearly mark out the three sections of the hex dump. The part of the dump that is of interest is found in the middle with hex numbers in groups of two. This middle section represents the actual frame. The offset numbers found in the first column indicate the position of each line in the dump. The java program uses this offset to find the start of the new frame with offset value "0000" and also to identify new lines. The last column is the ASCII translation of the hex numbers which is discarded. After knowing the format of the hex dump, the java program will parse out each line of hex numbers and convert them into binary, stringing each line

together to form one serial bit line seen in Figure 4.4. Finally, The 32 CRC bits that are found in the FCS field at the end of the frame is removed. Once each of the frames have been reformatted it is outputted to the "dataMatlabFormat.dat" and ready for use in the MATLAB/Simulink simulation. This java program code is provided in the Appendix B for further reference.

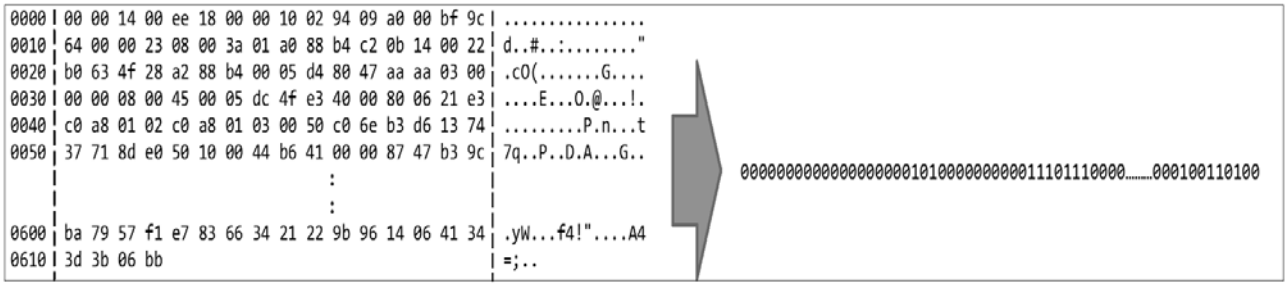


Figure 4.4 Frame Reformat from Wireshark to MATLAB/Simulink Example

Referring back to Figure 4.1, section B is the starting point in the MATLAB/Simulink simulation. This section is responsible for simulating side channel communication. As it was defined earlier in the background section of this thesis, a side channel is created by purposely corrupting frames using a different CRC polynomial. As it can be seen, there are only two CRC polynomials used here in the "General CRC Generator", one is termed as "Default" and the other as "Koopman". The Default CRC polynomial was used as the normal communication while Koopman was used as the side channel communication. The CRC generator function is designed to take a frame, calculate the CRC value based on the specified polynomial and add it to the end of the frame. Section B starts off at the decision gate indicated by point 1 in the figure where two things happen. The first thing is the values found in the "isFCSGood.dat" file described earlier is used to identify the

corresponding good and erroneous frames found in the "dataMatlabFormat.dat" file. Reiterating the fact earlier that this guarantees that the frame bits have not been naturally corrupted during capturing from section A. Any frames that were captured in error are discarded. The second thing is the "protoFile.dat" file is used to identify if the frame received in good was HTTP or FTP based on source port value 80 or 21 respectively. Once the frame has been identified as either HTTP or FTP it is pushed through the decision gate and on to the CRC generator. If the frame was identified as FTP then the CRC is calculated using the Koopman CRC polynomial to simulate a side channel frame. If the frame was identified as HTTP then the CRC is calculated using the Default CRC polynomial simulating a normal frame. As seen at point 2 is another CRC generator using the same Default CRC polynomial. The frame that goes through the generator at this point is the same one that was identified as good before it went into the decision gate at point 1. The reason for this second CRC generator using the Default polynomial is for generating the uncorrupted version of the same frame for comparison later when calculating the Hamming distance.

Section C commences at point 3 after the appropriate CRC is generated for the frame and appended to the end. The purpose of section C in the MATLAB/Simulink simulation portion is to simulate the channel properties which includes the probability for a frame being in error and if so, the number of bits flipped. At point 3 the frame is given a chance to be in error based on the probability inputted to the "Frame Error Probability Decider" function. At the decision gate at point 4, if the frame is determined to not be in error then it is sent to the next section D of the simulation. However, if the

frame is determined to be in error is it passed through the "AWGN" channel for corruption. The important thing to note here is the use of the SNR value that comes from the "SNRFile.dat" mention earlier to ensure that the number of bits flip in the frame is depended on dynamic values and not a fixed ones. This value is obtained for each corresponding frame and is used as input into the channel at point 5 to corrupt that frame and it is then sent to the next section D.

Section D has two responsibilities, the first one is to compare the two different frames received from point 2 and the frames that passed through the simulation from point 1 by calculating the Hamming distance at point 6. Note that the Hamming distance is calculated on the FCS portion of the frame otherwise known in this case the CRC value at the end of the frame. If the calculated Hamming distance value is 0 then it can be confidently identified that the frame is a normal frame that was not corrupted. Otherwise a Hamming distance value of greater than 0 has two possible cases. The first case is that it was a normal frame that was naturally corrupted or the second case is that it was side channel frame using the Koopman CRC polynomial. After the Hamming distance is calculated, the CRC value of the frame originating from point 1 and its resulting Hamming distance it is sent for output at point 7 which marks the second responsibility of section D. The output is also responsible for evaluating and printing the results. The output reports on the individual Hamming distance calculated for each received frame. It also calculates metrics such as the true positive(TP), false positive(FP), true negative(TN), false negative(FN), accuracy, sensitivity, specificity and the F-Score which is defined next in Table 4.1. These scores listed in the table will be reviewed in detail in

the next chapter when they are used in evaluating how effective the Hamming distance is as a metric for detection.

Table 4.1 Evaluation Scores and Metrics Definitions

	Evaluation Scores & Metrics	Definition
Score	**True Positive (TP)**	Side channel frames that are detected as side channel frames
	False Positive (FP)	Non-side channel frames detected as side channel frames
	True Negative (TN)	Non-side channel frames detected as non-side channel frames
	False Negative (FN)	Side channel frames that are detected as non-side channel frames
Metric	**Accuracy**	(TP+TN)/(TP+TN+FP+FN)
	Sensitivity/Recall	TP/(TP+FN)
	Specificity	TN/(TN+FP)
	Precision	TP/(TP+FP)
	F-Score	(2 * Precision * Recall)/(Precision + Recall)

This concludes the detailed overview of the testing environment which will be later utilised in the next section to validate the Hamming distance metric using a known Perceptron Learning algorithm and used for determining the Hamming distance threshold that can be used for detection of side channel.

Chapter 5

Validating and Evaluating the Hamming Distance Metric

So far Hamming distance has only been introduced as a possible detection metric for side channel communication. In order to be able to confidently say that it can be used as a detection metric, further validation is needed. Once the metric is validated, the next question that follows is how effective is this metric for detection. This question will be explored using well known evaluation metrics listed in Table 4.1 from the previous chapter. Before validation or evaluation can be performed, testing data must be obtained through some experiments which requires a detailed examination of the experimental setup.

5.1 Experimental Setup

In order to obtain data for use in the validation and evaluation of the Hamming distance, a physical experimental setup is needed to capture the necessary frames for the testing environment outlined in Chapter four. This testing environment had a real component that used the AirPcap and Wireshark together to capture real data frames for using in the simulation portion. The whole experimental setup is illustrated by Figure 5.1. The setup consists of 5 nodes that are communicating and one agent node that is in promiscuous

mode responsible for capturing data. As seen in the figure, the agent node is equipped with the AirPcap adapter and is running Wireshark. Being in promiscuous mode the agent is not participating in the communication and thus to avoid any chance of mistaken capture from the agents built-in wireless card, it was disabled. Each node in the setup were each given a static IP that started from 192.168.1.2 to 192.168.1.6.

For the purpose of referencing, each node will be identified by the far right least significant digit in its IP address. Node 2 with the label "Red Node" located in the top floor of the house is tasked to run both the HTTP and FTP server applications. The HTTP stream rate is 192kb/s which is approximately 23.4kB/s while FTP transfer rate is varied from 16kB/s, 32kB/s and 64kB/s. HTTP will stream a 22.5MB mp3 music file while FTP will transfer a 5MB file for all three varied rates. Node 3 also labelled with "Red Node" located in the basement floor will be the only node that will participate in the FTP transfer. As it was described in the previous chapter in the test environment section, the FTP traffic will be simulated as the side channel while the HTTP will be simulated as the non-side channel traffic. Node 2 and 3 will be participating in side channel and thus the reason why they are labelled as red nodes. Along with the side channel FTP traffic, nodes 2 and 3 will also participate in non-side channel traffic by streaming HTTP traffic. Nodes 4, 5, and 6 labelled with "Blue Node" will only participate by streaming non-side channel HTTP traffic. Note that the number of nodes was also varied from 5, 4, 3 and finally 2 to form different node configurations. Nodes 2 and 3 will always be left in the scenario as they are the only ones participating in side channel communication while other nodes are taken out. The order of removal was arbitrary selected which was the following:

1. *Node 6 (192.168.1.6)*
2. *Node 4 (192.168.1.4)*
3. *Node 5 (192.168.1.5)*

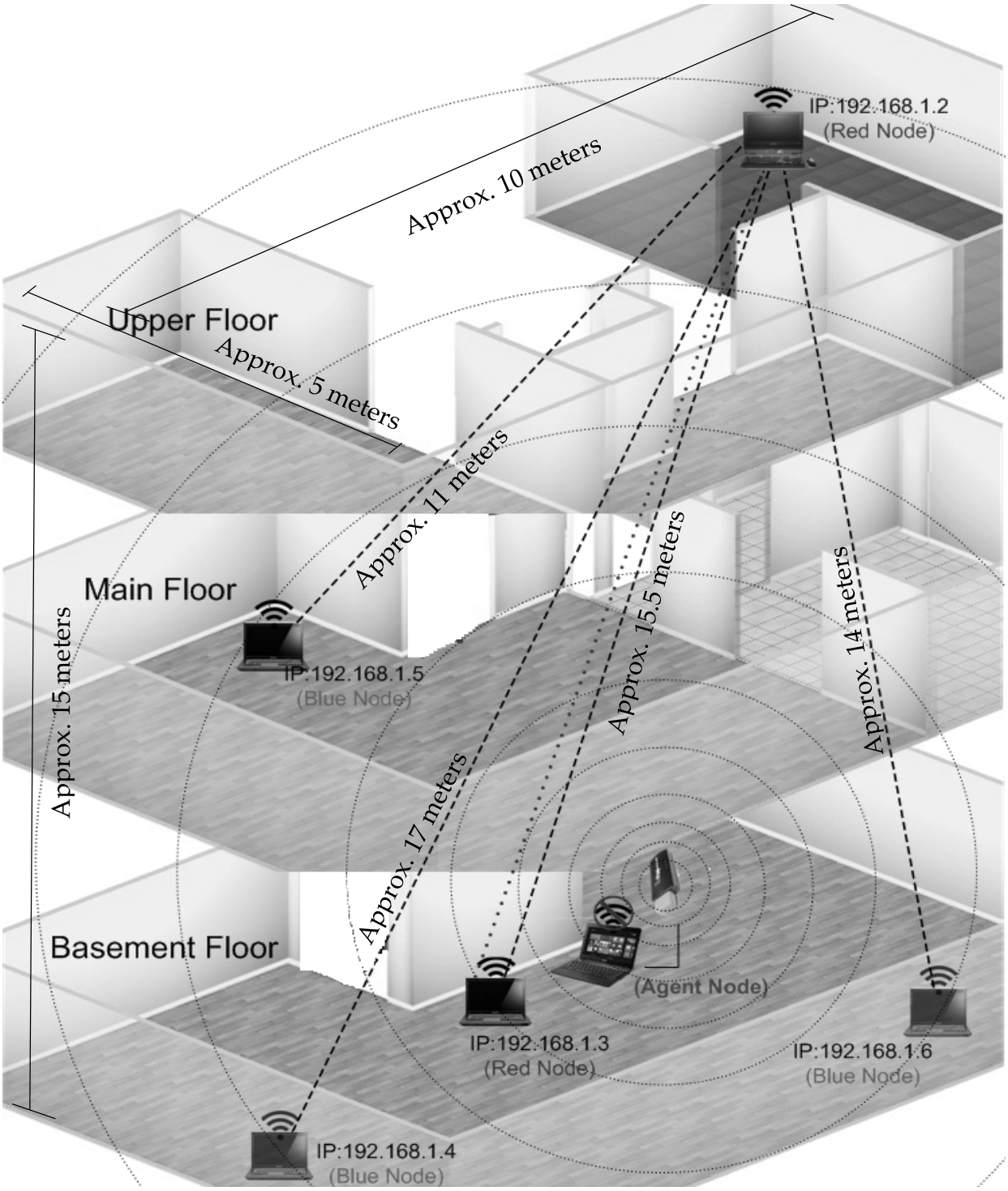


Figure 5.1 Experimental Setup for Capturing Real Data Frames

Essentially each node configuration had three different experiment runs for the different FTP transfer rates. One other variable that needs to be mention but not directly included in the node setup is the frame error percentage (FE %). This FE% is introduced in the simulation portion which was varied from 0% to 100% with 5% step increases. With all these parameters combined this would make up 252 total experimental scenarios. These parameters can be summarized in Table 5.1 below.

Table 5.1 Parameters List for Experimental Scenarios

Parameters	Parameter Values
Number of Nodes	5, 4, 3, 2
FTP Transmission Rates	16kB/s, 32kB/s, 64kB/s
Frame Error Percentage (%)	0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 ,55, 60, 65, 70, 75, 80, 85, 90, 95, 100

5.2 Validating Using the Perceptron Learning & Pocket Algorithm

To recap, the Hamming distance measure was the number of bits different between two CRC bit values. It was seen by the preliminary work done in chapter 3 that this metric was very promising for the use of detection based on the hypothesis. This hypothesis was that the Hamming distance value would be significantly higher for an intentionally corrupted frame (side channel frame) than a naturally corrupted frame when compared with the actual CRC value that was calculated on the same frame. The average Hamming distance values were approximately 16 versus 8 respectively which shows a noteworthy gap between the two that suggest the metric can be plugged into a classifier algorithm

and used for validation. In other words, if a known classifier algorithm is able to use this Hamming distance metric to find distinct clusters or separation between given data then it is safe to say that the metric is valid for detection. This will start the introduction into the Perceptron Learning Algorithm (PLA) and Pocket Algorithm (PA) for the validation of the Hamming distance metric.

Perceptron Learning Algorithm (PLA) is a supervised learning algorithm with reinforcements [27]. It is capable of learning how to classify data that is linearly separable [27]. The main idea is that it uses a simple line to separate the different clusters of data points. Each line that separates correctly the two different cluster is called a perceptron [27]. This line is based on the equation $y = mx + b$ where m is the slope and b is the intercept. This equation is rewritten as $w_1x + w_2y + w_3b = 0$ where b is equal to 1 and coefficients that make up the weight vector (w_1, w_2, w_3) [27]. Both training and testing data set consist of points with x, y pairings. However for each point in the training data, there will be an identifier that tells the algorithm which class it belongs to [27]. These classes can be thought of as clusters which can be identified by a 0 or 1 [27]. PLA uses these identifiers in the learning phase for reinforcement learning [27]. The learning phase starts by drawing a line with a random slope and intercept which are specified by randomly picking the weights [27]. Note that this random way of picking the weights is done only once to initialize them. Once the first line is drawn, the next step is to evaluate the position of the line from the data points [27]. Essentially this line would act as the divider between the two different classes. So for each data point (x,y) an error is calculated based on which side of the line the point falls on [27]. Based on the error PLA

will readjust its weight vector and redraw the lines. It will repeat these weight adjustments until the error converges to zero. An example can be seen in Figure 5.2 below.

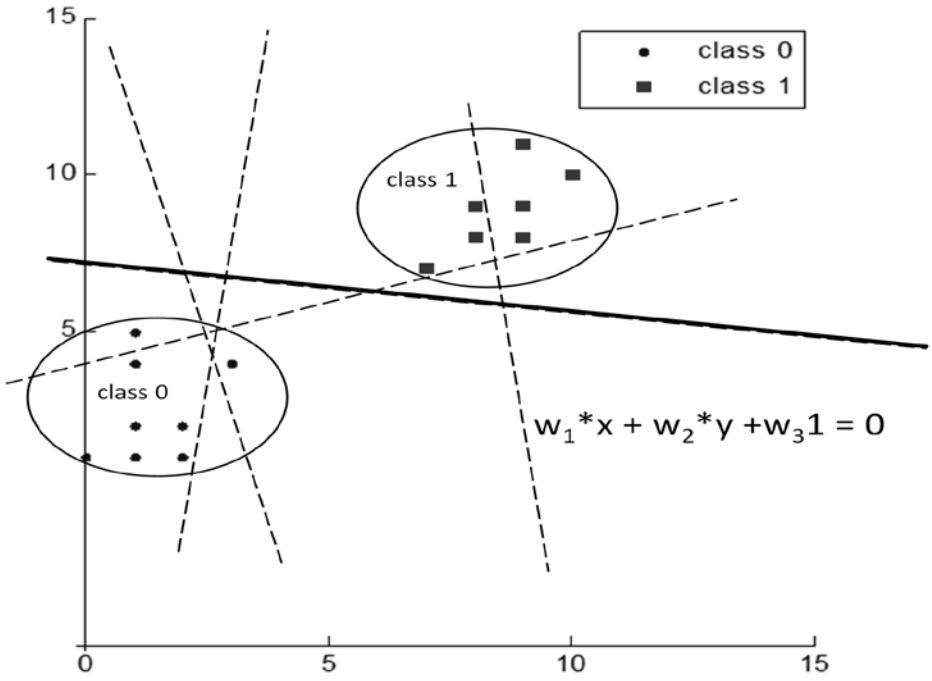


Figure 5.2 PLA Training Data Example

Figure 5.2 shows all the different possible perceptron that could be drawn that are identified by the dotted lines. The solid black line shows the possible final perceptron that completely separates the two different classes. Note that the number of perceptron that are obtained during any run of the PLA is always different and also the equation line of the final is not guaranteed to be the same. If the data is like the above example in Figure 5.2 then it can be characterized as linearly separable and the algorithm will eventually converge to zero. If the data is not linearly separable then the error will never converge

to zero. The worst case is that the error will start to increase or fluctuate based on the number of iteration it runs causing the algorithm to run infinitely. This is where the "Pocket" Algorithm (PA) comes into play. PA handles non-linearly separable data by changing two things. Firstly, because the data is not linearly separable which could cause the original Algorithm to run infinitely, PA requires that a predetermined max iteration value must be set [27]. Secondly, PA will keep track of the last weight vector values obtained after each iteration and compare them with the next. Based on the calculation for that iteration, the best weight vector is kept and the others are discarded [27]. This is like the analogy of taking the old one out of the "pocket" and replacing it with a better new one [27]. This process is done for each iteration until the maximum iteration is reached [27]. Once the training data is exhausted and the final perception is obtained then the testing may begin. Test points are then added to the system and classified based on the side of the perceptron line it falls on [27]. Both these algorithms are very simple to understand and implement which is one of the reasons why it is chosen to be used in the validation of the Hamming distance metric. The java code containing the implementation of the Pocket Algorithm can be found in Appendix C for further reference. In particular, PA is able to handle non-separable data which is a great advantage for dealing with real network data.

For validation in the Pocket Algorithm for validation, data was obtained from one of the 5 node scenarios outline in the experimental setup section. The 5 node scenario parameters are summarized in Table 5.2 below.

Table 5.2 Test Parameters for 5 Node Scenario (Validation using PA)

Test Parameters	Value
Number of Nodes	5
Number of Agents	1
Number of Red Nodes	2
Number of Blue Nodes	3
Types of Traffic	HTTP & FTP
Transfer Rates (HTTP)	23.4kB/s
Transfer Rates (FTP)	16kB/s
Number of side channel links	1 *(indicated by dotted line)*
Number of non-side channel links	4 *(indicated by dashed line)*
HTTP Stream File	22.5MB mp3 File
FTP File	5MB zip File
HTTP Stream Duration	Start: 0 seconds End: 480 seconds
FTP Transfer Duration	Start: 120 seconds End: 440 seconds
Frame Error %	0%-100% with 5% steps

The experiment runs as follows, at time 0 seconds all 4 nodes (3, 4, 5, and 6) will start the HTTP music streaming from node 2. The agent node will start capturing traffic data using Wireshark. A stopwatch is used to keep time. When the time hits 120 seconds (2minutes) the FTP transfer is started from node 3. At time approximately 440 seconds (7minutes) the FTP transfer is complete. The streaming continues on all 4 nodes until the full 480 seconds (8minutes) duration is completed, at which time the agent node's Wireshark capture is stopped and saved for preprocessing. This preprocessing and

simulation can be reviewed in Chapter four when the detailed overview of the hybrid testing environment was given. Once processed it is ready to be used in the simulation portion of the hybrid testing environment. The data will flow through the simulation as described earlier with the FE% varied from 0%-100% with the 5% step increase. This would yield a total of 21 different results. Once the simulation is finished the summary of the resulting frame counts are provided along with the Hamming distance output results. For each of the 21 results the Hamming distance values and both the naturally corrupted and intentionally corrupted (total counts listed below) are consolidated into one data file output. This output data is now used for input into the Pocket Algorithm for training. The testing data was obtained from a running the same 5 node scenario with the same parameters except that it was ran only for approximately 5 minutes 38 seconds and the file sent over FTP was started approximately after 1 seconds to ensure that the whole file was transmitted. The total counts for the testing data is also listed below. Both the training and the testing data was formatted to form a scatter plot where each Hamming distance was set as both x and y coordinate. After putting the training data into the Pocket Algorithm, the results can be seen below in Figure 5.3 plot.

List of Resulting frame counts from the simulation (PA-Training data):

- *Total Number of Corrupted frames: 190,486*
- *Total Number of naturally corrupted frames: 190,276*
- *Total Number of Intentionally corrupted frames (FTP/side channel): 210*

List of Resulting frame counts from the simulation (PA-Testing data):

- *Total Number of Corrupted frames: 20,686*
- *Total Number of naturally corrupted frames: 20,476*
- *Total Number of Intentionally corrupted frames (FTP/side channel): 210*

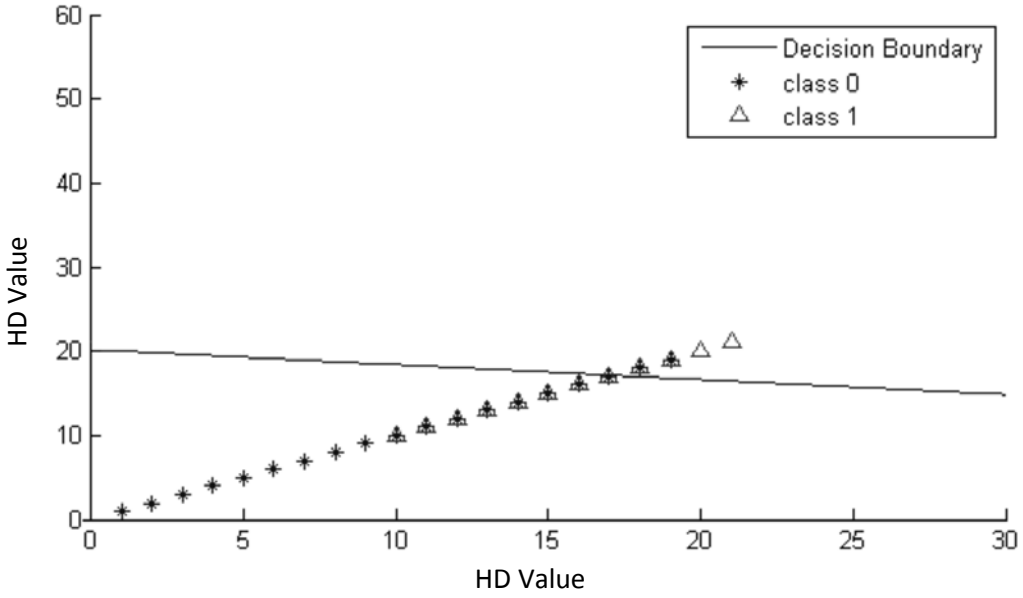


Figure 5.3 Training Data Scatter Plot of Frame's HD values and Perceptron Result

Figure 5.3 presents all the 20,686 Hamming distance values classified into the two classes 0 and 1. Class 0 represents the Hamming distance values obtained from non-side channel frames and class 1 are the values from the side channel frames. Note that the plot figure is a little deceiving as it looks like only a total or 31 data points are plotted including both class 0 and 1. The reason for this is the overlapping Hamming distance values that are plotted on top of each other. From Figure 5.3 it is clear that there isn't a distinct line

between the two classes which shows the non-linearity of the data. However a perceptron was still found. This perceptron can then be used in the testing phase with test data. The results of the testing phase can be seen below in Figure 5.4.

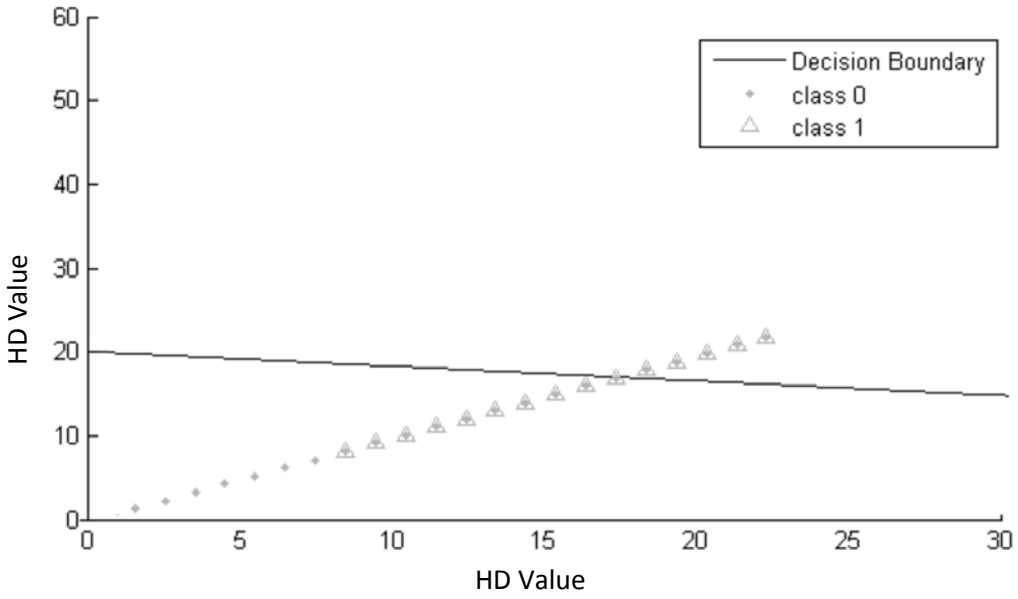


Figure 5.4 Scatter Plot of Frame's HD values and Test Data Results of using the obtained Perceptron

The results show from Figure 5.4 above that PA was able to correctly classify side channel from non-side channel frames approximately 99% of the time using the Hamming distance metric. However, it was only able to correctly classify side channel frames approximately 28% of the time. The lower 28% classification rate of side channel frames could be attributed to a higher amount of naturally corrupted frames versus the intentionally corrupted frames which was 20,476 and 210 respectively. This means that this lower 28% classification rate could be attribute to the higher volume of errors and

may yield a better result if a windowing analysis technique was applied in sampling only a small portion of frames that suspected to be side channel. Again this is left for future work that would explore the best window size for an effective analysis. Overall the results have shown that the Hamming distance is a valid metric for distinguishing side channel frames from non-side channel frames. Even though the Pocket Algorithm was able to show that the Hamming distance was effective in distinguishing between side channel and non-side channel frames, the question that still remains is how effective this metric is. This starts the exploration into the scores that can be used to evaluate how effective the Hamming distance metric is. These were briefly introduce in the previous chapter in Table 4.1.

5.3 Evaluation of the Hamming Distance Metric

In order to get an idea of why these scores were chosen, it is best to present the results to be evaluated. The experimental scenario used of the evaluation of the Hamming distance metric for this section also contains 5 node but is run with only one FE%. The FE% used was the actual value captured which was 18.87%. The 5 node scenario parameters can be summarized in Table 5.3 below. The resulting frame counts also follows below.

List of Resulting frame counts from the simulation (Evaluation 5 node scenario):

- *Total Number frames: 26,772*
- *Total Number of uncorrupted frames: 21,741*
- *Total Number of naturally corrupted frames: 5,031*
- *Total Number of Intentionally corrupted frames (FTP/side channel): 10*

Table 5.3 Test Parameters for 5 Node Scenario (Evaluation using F-Score)

Test Parameters	Value
Number of Nodes	5
Number of Agents	1
Number of Red Nodes	2
Number of Blue Nodes	3
Types of Traffic	HTTP & FTP
Transfer Rates (HTTP)	23.4kB/s
Transfer Rates (FTP)	16kB/s
Number of side channel links	1 *(indicated by dotted line)*
Number of non-side channel links	4 *(indicated by dashed line)*
HTTP Stream File	22.5MB mp3 File
FTP File	5MB zip File
HTTP Stream Duration	Start: 0 seconds End: 480 seconds
FTP Transfer Duration	Start: 120 seconds End: 440 seconds
Frame Error %	18.87

The results from this 5 node scenario outlined in Table 5.3 can be represented as a scatter plot of resulting Hamming distance values in Figure 5.5 below. The triangles are the FTP that represents side channel frames and the dots are the HTTP that represents non-side channel frames. From the figure it is clear to distinguish between the non-side channel frames that are in error from the ones that are not. The dots that are concentrated on the line where Hamming distance is 0 are all the frames that are not in error while all others scattered above are erroneous. Note also the region where the FTP transmission occurred

which is sectioned out by the drawn box. A magnified version is also provided to give a better idea of the spread of frames within that FTP region as the unmagnified version looks falsely like there are only 6 frames in a straight line. In order to get this magnified version, the x-axis scale was simply stretch out while the y-axis was left untouched .One other thing to note is that there is no clear separation between the Hamming distance values of the side channel frames and the non-side channel frames. This can be attributed to the natural error causing random number of bits to be flipped during transmission. If for example a detection threshold was used, depending on the where these overlapping points fall, they could account for the false positives and false negative defined in Table 4.1. If the detection threshold was picked where the Hamming distance was equal to 2 for example, from the figure it can be seen that any dots above this threshold would be considered as false positives. At threshold equal to 14 yields fewer false positives but also causes false negatives with a few of the triangle side channel frames below the threshold. Choosing a threshold like 17 for example would yield no false positives but again increase the number of false negatives as more triangles are seen below the threshold line. This alludes to the idea that the effectiveness of using Hamming distance for detection is based on the threshold value chosen. By varying the Hamming distance values, the effective threshold for detection can be determined when the ideal condition is met where the both the true positive and true negative scores are maximized while both the false positive and false negative scores are minimized. This would require the true and false scores to be calculated for each possible threshold and the threshold that yields the desired scores is chosen.

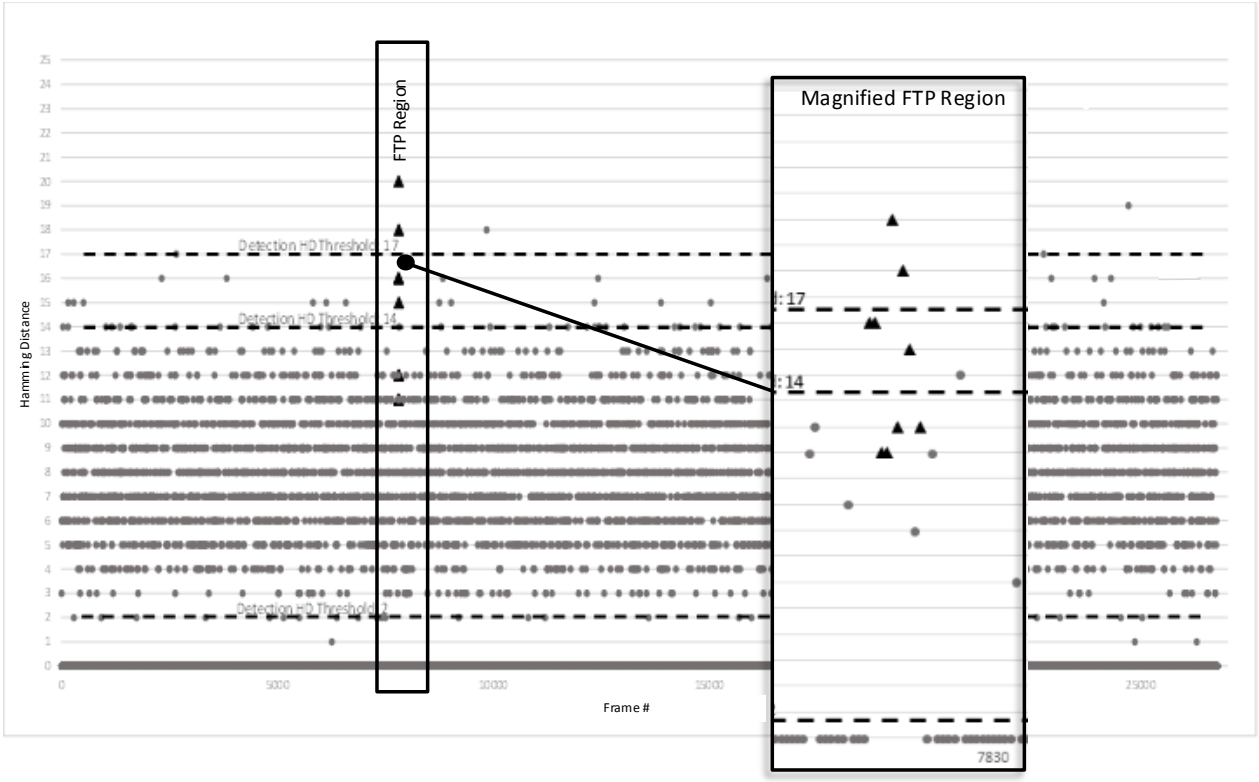


Figure 5.5 Hamming Distance Results for Scenario:5MB@16kB/sec with 18.87% Frame Error

This can be challenging if there is a large number of scenarios which could introduce human error from possible inconsistencies when evaluating the four scores for the best threshold. To avoid this, five more scores can be used that are based on the four true and false scores. These are also listed in Table 4.1 as Accuracy, Sensitivity/Recall, Specificity, Precision and F-Score. These evaluation metrics are commonly used in evaluating and assess classification and detection algorithms [28]. They can be used to further assess the Hamming distance metric for its effectiveness for distinguishing side channel from non-side channel frames. Accuracy assesses the overall effectiveness of an algorithm by measuring the total number of true positives and true negatives versus both the false

scores [28]. Sensitivity or recall assesses the effectiveness of the algorithm to correctly identify true positives [28]. Specificity assess the effectiveness for the algorithm to correctly identify true negatives [28]. Precision assesses the algorithms predictive power in correctly identifying the true positives from the false positives [28]. The F-Score is a composite measure that focuses on the effectiveness of the algorithm to identify true positives compared to the false positive and false negative. To recap with respect to detection of side channel frames using Hamming distance, accuracy will measure how well Hamming distance can correctly identify side channel and non-side channel frames. Sensitivity or recall will measure how well Hamming distance can correctly identify all instances of side channel frames. Specificity on the other hand will measure how well Hamming distance can correctly identify all instances of non-side channel frames. Precision will measure how well Hamming distance can predict the side channel frames from the non-side channel frames. Finally the F-Score measures how well Hamming distance can correctly identify side channel frames while limiting the false identification such as non-side channel frames identified as side channel or side channel frames missed identified as non-side channel. Looking at the formula of the F-Score from Table 4.1, it is apparent that it is based on both precision and sensitivity/recall which depend on the true positive, false negative, and false positive scores. Taking this into account, a high F-Score will indicate that the Hamming distance was able to effectively identify side channel frames while at the same time limiting false scores such as non-side channel frames identified as side channel (False Positive) or side channel frames identified as non-side channel (False Negative).Ultimately, this F-Score metric will be used to assess the effectiveness of the Hamming distance metric at different threshold values.

Taking the results from the 5 node scenario presented in Figure 5.5, all the evaluation metrics can be computed for each of the Hamming distance threshold values from 1 to 21. Note that the only computed values are for the first 21 threshold as there are no points above this value. Also Note that the time window used was the full eight minute run of the scenario. This means that the resulting evaluation metrics are based on the entire data set represented in Figure 5.5 and not only within the "FTP Region" when the FTP was transmitted. The selection of a good time window to use for detection would easily be a thesis problem on its own and is not within the scope of this thesis but suggests future work that can be done. The advantage of using the entire resulting data set is it will add additional noise and will act as a stressor for testing. This will help strengthen the results of the effectiveness of Hamming distance metric. The resulting calculations can be seen in Table 5.4.

Table 5.4 Calculated Evaluation Metric Results for 5node@16kB/s with 18.87% FE

Threshold	True(+)	False(+)	True(-)	False(-)	Accuracy	Sensitivity	Specificity	Precision	F-Score
1	10	5027	21735	0	0.8122	1.0000	0.8122	0.0020	0.0040
2	10	5001	21761	0	0.8132	1.0000	0.8131	0.0020	0.0040
3	10	4909	21853	0	0.8166	1.0000	0.8166	0.0020	0.0041
4	10	4696	22066	0	0.8246	1.0000	0.8245	0.0021	0.0042
5	10	4284	22478	0	0.8400	1.0000	0.8399	0.0023	0.0046
6	10	3681	23081	0	0.8625	1.0000	0.8625	0.0027	0.0054
7	10	2932	23830	0	0.8905	1.0000	0.8904	0.0034	0.0068
8	10	2104	24658	0	0.9214	1.0000	0.9214	0.0047	0.0094
9	10	1360	25402	0	0.9492	1.0000	0.9492	0.0073	0.0145
10	10	772	25990	0	0.9712	1.0000	0.9712	0.0128	0.0253
11	8	400	26362	2	0.9850	0.8000	0.9851	0.0196	0.0383
12	6	196	26566	4	0.9925	0.6000	0.9927	0.0297	0.0566
13	6	81	26681	4	0.9968	0.6000	0.9970	0.0690	0.1237
14	6	31	26731	4	0.9987	0.6000	0.9988	0.1622	0.2553
15	**5**	**13**	**26749**	**5**	**0.9993**	**0.5000**	**0.9995**	**0.2778**	**0.3571**
16	2	5	26757	8	0.9995	0.2000	0.9998	0.2857	0.2353
17	2	3	26759	8	0.9996	0.2000	0.9999	0.4000	0.2667
18	1	1	26761	9	0.9996	0.1000	1.0000	0.5000	0.1667
19	1	0	26762	9	0.9997	0.1000	1.0000	1.0000	0.1818
20	0	0	26762	10	0.9996	0.0000	1.0000	NaN	NaN
21	0	0	26762	10	0.9996	0.0000	1.0000	NaN	NaN

As stated before, the F-Score is affected by precision and sensitivity/recall and it's at its highest when both of these metrics are maximized. This is confirmed in the results where both precision and sensitivity/recall are the highest at 0.2778 and 0.5000 respectively, yielding the best F-Score of 0.3571. Looking deeper into the definition of precision and sensitivity/recall they are affected by the True positives scores. The difference is that sensitivity focuses on the ratio of true positive versus false negative and

precision on true positive versus false positives. So this means that sensitivity/recall only evaluates how well side channel frames are classified for a given threshold value but does not take into account the number of errors in missed classifications of non-side channel frames. This is where precision comes into play by accounting for these false positives (non-side channel frames missed classified as side channel). This can be seen from the result Table 5.4, where thresholds with Hamming distance of 1 through 6 yields a perfect effective sensitivity/recall score of 1. However the corresponding precision score is closer to 0 (0.0020 – 0.0027) which gives a better idea of the effectiveness of those thresholds. In other words because the thresholds for detection is so low anything above it is considered as side channel frames which explains the high amounts of false positives attributing to the precision score of nearly 0. This can be verified visually by looking at Figure 5.5 for example, if a Hamming distance threshold was picked at 2, any erroneous non-side channel frames (indicated by the dots) falling above this line would be counted as side channel which would add to the false positives score. Conversely, if the Hamming distance threshold was picked at 17 for example, from the results the corresponding sensitivity/recall and precision values are 0.2000 and 0.4000 respectively. This is almost the reverse effect seen here, where the precision may falsely indicate a better effective threshold but in actuality by looking at the sensitivity/recall metric it is not. This is due to the fact that the false positives have dramatically decreased and the false negatives have increased. Again visually this can be seen from Figure 5.5 where the majority of non-side channel frames correctly fall below the threshold at Hamming distance 17. At the same time some side channel frames do fall below this threshold value and therefore are classified incorrectly as non-side channel adding to the number of false negatives.

After examining these two extremes, it is evident that the ideal scores for precision and sensitivity/recall when both of them are maximized at the same time. Instead of trying to weigh these scores independently, the F-Score is used instead as an overall indicator for the most effective Hamming distance threshold value. As for the other evaluation metrics such as accuracy and specificity, they both give a good indication on how well each of the threshold value can correctly distinguish side channel from non-side channel frames up to a certain point. This point is when the false negative start to become more evident and then these measures just plateau at a value. These two measures are good to confirm directionally how effective each threshold is but the F-Score appears to be the best indicator to find the optimal one. As it was seen from the resulting table above, the most effective Hamming distance threshold value that resulted from this scenario was 15 which yields an F-Score of 0.3571. These results help show that even with all the noise an effective Hamming distance threshold value is still attainable. The question that still remains is if an effective threshold value can still be found with other scenarios. This will be explored next.

Chapter 6

Determining an Effective Threshold

So far it has been shown that the Hamming distance has very promising results in its validation and evaluation. Presented with the 5 node scenario used in the evaluation section of Chapter 5, an effective threshold of Hamming distance equal to 15 was obtained with an F-Score of 0.3571. The only thing that could be said about this result is that an effective Hamming distance threshold was found for the 5 node scenario with parameters outline in Table 5.3. In other words there needs to be some confidence that these results did not occur by chance due to some unknown variable in this one 5 node scenario.

6.1 Hamming Distance Population Mean Confidence Interval

One way to do this is to run as many scenarios as possible in order to see if these results can be generalized to other circumstances. This can be impossible as there may be an infeasible number of scenarios to run. A finite way would be to generate all possible frame bit strings instead of actually capturing real data and passing them through the simulation. Through the simulation, the corresponding CRC values are generated for each frame using both the default and the Koopman CRC32 polynomial. Finally, the total average Hamming distance can then be determined for both non-side channel and side channel frames. If there is a statistically large difference between the two population averages, then it can be 100% stated that a threshold for detection would be in between

the two averages. The only issue is that in order to obtain whole population of all possible combination of a 2342 bit frame [11], there would have to be in total 2^{2342} frame bit strings generated. To get an idea of what 2^{2342} looks like, Maple 18 [29] was used to calculate this exponential number which resulted in small paragraph of digits.

This could present itself as future work in determining all the possible 2^{2342} combination frame bit strings but due to the limited time and computational power another option is to come up with an educated guess on what the average Hamming distance would be if it were possible. In other words find some confidence level that predicts the mean population Hamming distance. This can be done by finding a confidence interval that the mean population would fall in with a suitable confidence level using a large sample of the population's 2^{2342} possible frame bits. Using the same testing methodology, a sample of these frames can be obtained from the capturing of real data. The only thing to watch out for is that the Hamming distance values obtained are normally distributed and the sample size is large enough (>30), in order to obtain a confidence interval using the z-tables [30].

Let's first re-explore the results obtained from the 5 node scenario outlined in Table 5.3 used for the evaluation section of Chapter 5. The same results are used to calculate the confidence intervals for the mean Hamming distance value for both non-side channel and side channel frames. The purpose is to first predict what the population Hamming distance mean for both non-side channel and side channel frames is and secondly, observe if they fall below and above the determined threshold. In other words if the

predicted population Hamming distance mean for both the non-side channel frames and side channel frames are below and above the threshold respectively. The first thing that has to be done is take out all the uncorrupted non-side channel frames that correspond to all Hamming distance values of 0. Let's first calculate the mean, mode, median, sample variance, standard deviation, coefficient of variation, Kurtosis and Skewness for the Hamming distance values obtained from all the corrupted non-side channel frames. Remember that these are the frames that are calculated using the default CRC32 polynomials that have been naturally corrupted. Referring to plot in Figure 5.5, these are all the dots that have a Hamming distance value greater than 0. The resulting statistical result and distribution can be seen in the following Figure 6.1.

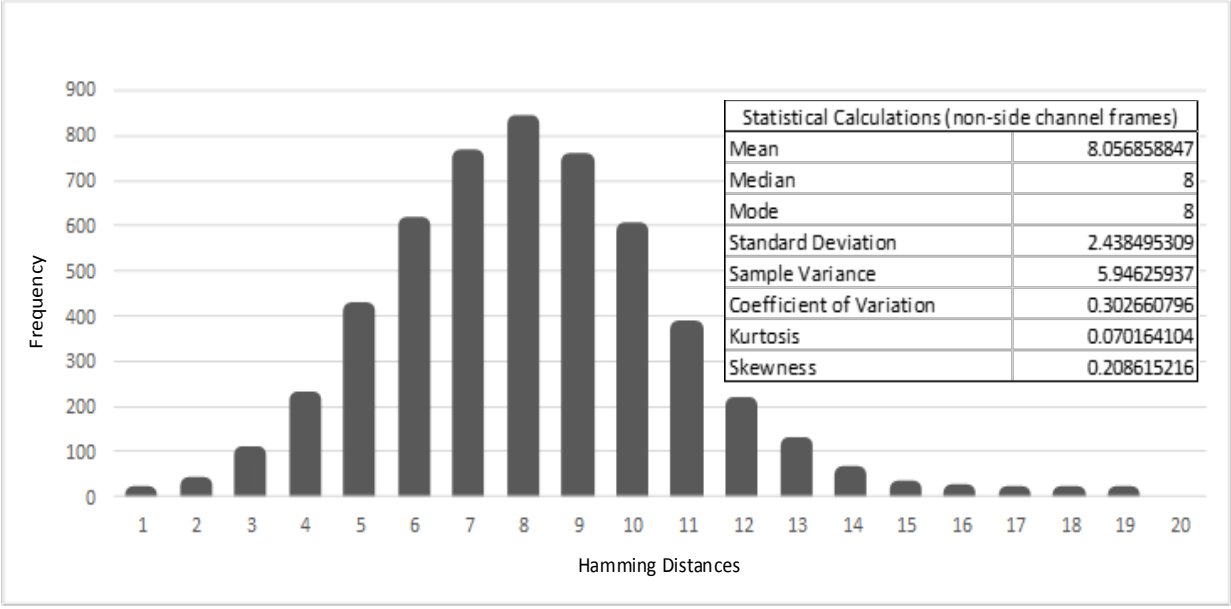


Figure 6.1 Hamming Distance Value Histogram Distribution & Statistical Calculation for Scenario: 5node@16kB/s with 18.87% FE

As it can be seen from Figure 6.1, the median and the mode is 8 which is close to the mean Hamming distance of approximately 8.057. The coefficient of variation is approximately 0.303 which by definition shows that the Hamming distance values are pretty uniform and centered on the mean. As expected it can be observed that the shape of the histogram plot is normally distributed. This normality can also be confirm from the Kurtosis and Skewness values which lay between -1.96 and +1.96 [30]. Now the sample mean and the standard deviation can be used to calculate the mean confidence interval with a particular confidence level. The resulting mean confidence interval for both 95% and 99% confidence level can be seen below in Figure 6.2 below. What the results mean is that there is a 95% chance that the population mean Hamming distance will fall approximately between 7.989 and 8.124 and also a greater 99% chance that it's between 7.968 and 8.145. A 99% chance that the average Hamming distance value for any non-side channel frame is very close to 8 means that the resulting threshold of 15 determined by the best F-Score for this 5 node scenario is very promising as a detection threshold. The only thing left is to find out what the population mean Hamming distance is for side channel frames with the same level of confidence. Remember that these are the frames that were intentionally corrupted using the Koopman CRC32 polynomial. Referring to plot in Figure 5.5, these are all the triangles near the top.

Confidence Level=95%		*Confidence Level=99%*	
Confidence Level(95.0%)	0.067404759	Confidence Level(99.0%)	0.088597175
Lower bound mean	7.989454088	Lower bound mean	7.968261672
Upper bound mean	8.124263606	Upper bound mean	8.145456022

Figure 6.2 Population Mean Confidence Interval of Non-side Channel Frames for Scenario: 5node@16kB/s with 18.87% FE

One thing to note is that the number of side channel frames is only 10 which is definitely less than 30. This will mean that the distribution is not guaranteed to be normal as the sample size is too small to make that assumption [30]. However the confidence interval can still be computed using the t-distribution along with the t-table [30]. The resulting statistical and confidence interval calculation can be seen below in Figure 6.3. Notice first that resulting confidence interval has widened due to the sample size being less than 30. However results still show a 95% chance that the average population Hamming distance will fall approximately between 12.487 and 16.912 and also a 99% chance that it will be between 11.521 and 17.879. Even though the lower end of the mean falls below the threshold of 15, it is still significantly larger than the population mean of the non-side channel frames which indicates that there is separation between them. Obviously these confidence intervals are not as impressive as the once for non-side channel frames due to the small sample size. In order decrease the interval size but keep the confidence level at 95% and 99%, the sample size of side channel frames must be increased to at least 30.

Statistical Calculations (side channel frames)	
Mean	14.7
Median	15.5
Mode	16
Standard Deviation	3.093002856
Sample Variance	9.566666667
Coefficient of Variation	0.210408358
Kurtosis	-1.007001594
Skewness	0.238258363

Confidence Level=95%	
Confidence Level(95.0%)	2.212600953
Lower bound mean	12.48739905
Upper bound mean	16.91260095

Confidence Level=99%	
Confidence Level(99.0%)	3.17864264
Lower bound mean	11.52135736
Upper bound mean	17.87864264

Figure 6.3 Statistical Calculation & Population Mean Confidence Interval of Side Channel Frames for Scenario: 5node@16kB/s with 18.87% F_E

In order to increase the sample size of side channel frames, another 5 node scenario was ran with the parameters listed in Table 6.1. This was the same 8 minute capture with an increase in the size of the FTP file. In order to keep the capture with 8 minutes the transmission rate was increased to 64kB/s. Again the data will flow through the simulation as described earlier. The only thing to note is that the frame error probability value used for input into the "Frame Error Probability Decider" was 19.82%. This value was the true value obtained from the actual capture. The following list is the resulting frame counts:

List of Resulting frame counts from the simulation:

- *Total Number of frames: 26,461*
- *Total Number of uncorrupted frames: 21,277*
- *Total Number of naturally corrupted frames: 5,184*
- *Total Number of Intentionally corrupted frames (FTP/side channel): 37*

Table 6.1 Experimental Parameters for New Scenario: 5Node@64kB/s with 19.82% FE (Increased Side Channel Frames)

Test Parameters	Value
Number of Nodes	5
Number of Agents	1
Number of Red Nodes	2
Number of Blue Nodes	3
Types of Traffic	HTTP & FTP
Transfer Rates (HTTP)	23.4kB/s
Transfer Rates (FTP)	64kB/s
Number of side channel links	1 *(indicated by dotted line)*
Number of non-side channel links	4 *(indicated by dashed line)*
HTTP Stream File	22.5MB mp3 File
FTP File	18MB video File
HTTP Stream Duration	Start: 0 seconds End: 480 seconds
FTP Transfer Duration	Start: 90 seconds End: 454 seconds
Frame Error %	19.82

The results from this scenario outlined in Table 6.1 yielded a threshold value of also 15 with an F-score value of 0.5507. For further reference see Appendix D for the resulting evaluation metrics for this scenario. The confidence intervals for both non-side channel and side channel were calculated again for this new scenario. The results can be seen in Figure 6.4 and 6.5. Figure 6.4 shows both the statistical and confidence interval results of a non-side channel frames. It can be noted that the results are almost identical to the first 5 node scenario presented and more importantly the confidence intervals are pretty much

the same for the same 95% and 99% confidence level. In fact the interval is smaller than what was observed in the results shown in Figure 6.2. This is very significant because it confirms that due to the larger sample size, the confidence interval is smaller and falls within the interval calculated earlier. This is going to be expected for the side channel frames Hamming distance population mean as well because the sample size was increase to be greater than 30 frames. This is confirmed with the results shown in Figure 6.5 where the interval has become smaller for the same 95% and 99% confidence level. Showing a 95% chance that the average population Hamming distance will fall approximately between 14.760 and 16.862 and also a 99% chance that it will be between 14.401 and 17.220. Notice also that the upper bounds are pretty much the same as before but the lower bound mean has increased closer to15. These results help to show that non-side channel frames Hamming distance tend to fall closer to 8 while the side channel fall closer to 15. This also helps generalize the results to other types of scenarios as the frames that are captured are just samples of the 2^{2342} possible frames in the population. Essentially there is a 99% chance that the average non-side channel frames will fall somewhere between 7.929 and 8.106 and side channel frames falling between 14.402 and 17.220.

Statistical Calculations (non-side channel frames)	
Mean	8.017208043
Median	8
Mode	7
Standard Deviation	2.472492977
Sample Variance	6.113221521
Coefficient of Variation	0.308398256
Kurtosis	0.029096992
Skewness	0.173470475

Confidence Level=95%	
Confidence Level(95.0%)	0.067399328
Lower bound mean	7.949808715
Upper bound mean	8.084607371

Confidence Level=99%	
Confidence Level(99.0%)	0.088589698
Lower bound mean	7.928618345
Upper bound mean	8.105797741

Figure 6.4 Statistical Calculation & Population Mean Confidence Interval of Non-side Channel Frames for Scenario: 5Node@64kB/s with 19.82% FE

Statistical Calculations (side channel frames)	
Mean	15.81081081
Median	16
Mode	14
Standard Deviation	3.152052575
Sample Variance	9.935435435
Coefficient of Variation	0.19936059
Kurtosis	-0.582801877
Skewness	0.302170022

Confidence Level=95%	
Confidence Level(95.0%)	1.050946652
Lower bound mean	14.75986416
Upper bound mean	16.86175746

Confidence Level=99%	
Confidence Level(99.0%)	1.409221302
Lower bound mean	14.40158951
Upper bound mean	17.22003211

Figure 6.5 Statistical Calculation & Population Mean Confidence Interval of Side Channel Frames for Scenario: 5Node@64kB/s with 19.82% FE

It is clear to see from these scenarios the number of side channel frames affects both the confidence interval and the F-Score values. In these cases both the values are better. It will be advantages to examine what happens to these scores if the side channel frames are dramatically increased. The expected outcome would be that they would yield better results. In order to obtain more simulated side channel frames from the already existing captured data of the 5 node scenario at 16kB/s, the FTP data frames from port

number 51434 was filtered out this time to use in the simulation. The resulting frame counts can be shown as follows:

List of Resulting frame counts from the simulation using port 51434:

- *Frame Error percentage: 15.87%*
- *Total Number of frames: 33,175*
- *Total Number of uncorrupted frames: 27,975*
- *Total Number of naturally corrupted frames: 5,200*
- *Total Number of Intentionally corrupted frames (FTP/side channel): 6,413*

It can be seen from the resulting frame count that the FTP/side channel frames have increased to 6,413. This is approximately 20% more than the naturally corrupted ones. Realistically the number of side channel frames should not outnumber normal traffic within a specified time window to avoid detection but for the sake of testing this is allowed. It is clear that the FTP frames filtered out from port 51434 are the FTP data frames itself. Port 21 were FTP control frames while port 51434 were FTP data frames which explains the large difference in counts. In order to use the newly obtained FTP data frames from port 51434 for comparison of confidence interval and F-Score, the means must be confirmed that they come from the same population as the sample of frames from port 21. To confirm this the t-test is applied with the following hypothesis to test:

- ***Null Hypothesis**: The two means are equal (from the same population)*
- ***Alternative Hypothesis**: The two means are not equal (not from the same population)*

The results of the t-test show the t-statistic value to be -0.351 and the t-Critical value is 1.645. The null hypothesis is rejected based on the condition that if |t-statistic| > t-Critical value [31]. For this case the t-statistic value is less than the t-Critical value and thus there is not enough evidence to reject the null hypothesis. This means that there is a very good chance that they are equal which in turn can be stated that the sample comes from the same population. Intuitively this is true as the frame bits obtained would come from the same population of 2^{2342} possible frame bits. Now the confidence interval of the HD population mean can be calculated to see the effects of dramatically increasing the number of side channel frames for the same 95% and 99%.

The resulting mean confidence interval for non-side channel frames for both 95% and 99% are 7.952 ± 0.0759 and 7.952 ± 0.0998 respectively. Notice that this is again nearly identical to the confidence interval calculated for the other two 5 node scenarios with less side channel frames. The resulting mean confidence interval for side channel frames for both 95% and 99% are 15.974 ± 0.0797 and 15.974 ± 0.1047 respectively. Essentially there is a 99% chance that the average non-side channel frames will fall somewhere between 7.852 and 8.052 and side channel frames falling between 15.870 and 16.079. These results help to confirm that the means obtained from this larger sample of side channel frames has a smaller margin of error with the same 95% and 99% confidence interval. It also helps to confirm that there is still a threshold for detection that exist as there is clear separation between the two population means. The resulting F-Score value was 0.946 with corresponding best HD threshold for detection set at 11. Note that these results are

presented and can be confirmed in Appendix E. It is significant to note that the F-Score value is a lot higher when the number of side channel frames is also very high. This can be confidently attributed to the increase in side channel frames as the non-side channel frames are pretty much the same across all the 5 node scenarios evaluated. Again this confirms that realistically an adversary would have to dramatically reduce the number of side channels to successfully evade detection. The only question now is what threshold would be the most effective with a more realistic low side channel frame count.

6.2 Determining an Effective Threshold for Detection

Now that it has been confirmed that a Hamming distance threshold for detection does exist, the search becomes finding the most effective one to use across different types of scenarios. The first step in determining a generalizable solution that can be applied to other scenarios is figuring out what scenario variables that can affect the Hamming distance values. Restated, the question would be what possible variables in the scenario is this Hamming distance metric sensitive to? As it was shown before in Chapter five, the F-Score is the best indicator of how effective a Hamming distance threshold value is for a particular scenario. It was also shown that this F-Score value was greatly affected by the precision and sensitivity/recall metrics which in turn were greatly affected by the false negative and false positive scores. In a perfect scenario these scores would be 0 making precision and sensitivity/recall both 1 which in turn make the F-Score 1. So the question is what factor makes a scenario become flawless in order obtain a perfect F-Score? The answer to this question is having no error. Conversely stated, the amount of error affects the F-Score. This affect was also seen when the Hamming distance metric was used in the

validation using the Pocket Algorithm. The resulting threshold then was roughly 17 but had a 28% detection rate of side channel frames. This low percentage of detection was attributed to the high volume of errors compared to the number of side channel frames present which was used for training and testing. If this were the case then the two resulting Hamming distances would be 0 and values greater than 0. The values greater than 0 would identify all the side channel frames with no false negative or false positives. On the other hand when the number of side channel frames outnumbered the non-side channel frames the F-Score yielded better values that were closer to 1. Based on the characteristics of wireless communication and more specifically the MANET environment, it is clear that transmission error cannot be avoided and realistically the number of side channel frames would be very low to avoid detection.

However it is possible to identify scenario variables that could directly affect the amount of error which can then be varied to test for an effective Hamming distance threshold value. The scenario variables that may have an effect on the amount of error were introduced earlier in Chapter five when the experimental setup was discussed, are number of nodes, percentage of errors and the transmission rate. The number of nodes can affect the amount of errors due to radio interference or can increase the amount traffic in the shared medium which can cause more chances of error. If the cause of error is not known then the percentage of frame error can be introduced to see if an effective threshold is still obtainable. Seeing as the objective is to identify intentionally corrupted from naturally corrupted frames, the increase in error would cause the false negative and false positives scores to go up which cause the F-Score to go down. The transmission rate

will affect the number of errors seen at a particular time window. As there is no time window analysis, this variable maybe not affect the scores in this case as the HTTP rates are fixed and the FTP finishes transmission within an 8 minute window. However the rates will still be varied to see if there would be any effect.

To recap, the number of nodes was varied from 2, 3, 4 and 5 plus 1 agent in each. For each number of nodes the percentage of frame error was changed from 0% to 100% with 5% step increases. The transmission rate was be increased from 16kB/s, 32kB/s and 64kB/s. The total number of experiments was 252 (number of nodes * transmission rate * varied FE%). The summary of the parameters can be references in Table 5.1 in Chapter 5. After each experiment the most effective threshold is determined by the highest F-Score. Then the most frequently seen threshold value is compared with the population mean intervals determined earlier, to see where it falls. For example, if the threshold value falls within the Hamming distance population mean interval of the non-side frames then it might not be very effective. The ideal value would fall outside but below the population mean interval of the side channel frames. The results is presented in Tables 6.2 and 6.3.

Table 6.2 Highest/Best F-Score Value for each 252 Experimental Scenarios

	16kB/sec				32kB/sec				64kB/sec			
FE %	2 Nodes	3 Nodes	4 Nodes	5 Nodes	2 Nodes	3 Nodes	4 Nodes	5 Nodes	2 Nodes	3 Nodes	4 Nodes	5 Nodes
0	1	1	1	1	1	1	1	1	1	1	1	1
5	0.7826	0.7826	0.75	0.5556	0.7619	0.9091	0.7059	0.6957	0.7273	0.6667	0.8235	0.6364
10	0.6364	0.5455	0.5333	0.5	0.6667	0.8571	0.5217	0.6	0.6957	0.5556	0.6364	0.7368
15	0.5556	0.6667	0.7059	0.5714	0.7059	0.72	0.5333	0.5185	0.5833	0.6	0.4848	0.7273
20	0.5882	0.5882	0.7059	0.5217	0.5882	0.8182	0.4444	0.6667	0.6667	0.6	0.5263	0.7619
25	0.6316	0.4706	0.6667	0.3704	0.7059	0.72	0.5714	0.4138	0.5714	0.4615	0.5	0.5926
30	0.5	0.5333	0.6667	0.4444	0.7368	0.5833	0.5263	0.375	0.4516	0.5	0.6316	0.48
35	0.7	0.4706	0.6316	0.4211	0.6667	0.72	0.4706	0.3571	0.6	0.5333	0.48	0.5333
40	0.4	0.4545	0.75	0.3158	0.6316	0.6667	0.5714	0.5714	0.75	0.4706	0.3333	0.5833
45	0.4	0.5556	0.6667	0.3077	0.6667	0.6667	0.4762	0.3429	0.4444	0.7368	0.381	0.5
50	0.4615	0.4762	0.5333	0.3333	0.5385	0.6	0.4444	0.2353	0.625	0.625	0.381	0.4737
55	0.4615	0.4348	0.6667	0.2	0.6667	0.6667	0.4167	0.3333	0.4706	0.5714	0.4286	0.5333
60	0.5714	0.4	0.5882	0.5	0.4444	0.5333	0.381	0.3333	0.625	0.5556	0.4211	0.7778
65	0.5263	0.5	0.7059	0.3636	0.5263	0.6429	0.4444	0.1818	0.3529	0.5	0.2143	0.4706
70	0.6667	0.45	0.5714	0.2353	0.5217	0.5185	0.6087	0.5	0.625	0.48	0.5882	0.5
75	0.3226	0.5714	0.4706	0.4444	0.4444	0.7407	0.5	0.2727	0.4762	0.3571	0.2609	0.5556
80	0.4	0.3	0.5	0.1818	0.2667	0.6667	0.24	0.3043	0.4848	0.6667	0.5	0.6316
85	0.6667	0.5556	0.5263	0.3077	0.5556	0.4	0.6364	0.1818	0.4	0.4	0.3333	0.7059
90	0.5	0.4545	0.3529	0.375	0.3429	0.5	0.2069	0.2857	0.3077	0.6316	0.3077	0.7
95	0.6	0.5333	0.4	0.3478	0.625	0.6364	0.4615	0.1538	0.4444	0.5714	0.4706	0.3448
100	0.75	0.5545	0.4545	0.3333	0.5714	0.6316	0.4286	0.4242	0.4615	0.5714	0.2857	0.6316
Mean:	*0.556055*	*0.514895*	*0.59233*	*0.381515*	*0.5817*	*0.659895*	*0.47949*	*0.387365*	*0.538175*	*0.552735*	*0.449415*	*0.593825*
Median:	*0.5635*	*0.51665*	*0.6099*	*0.367*	*0.6066*	*0.6667*	*0.4734*	*0.35*	*0.5281*	*0.5635*	*0.4496*	*0.58795*
Variance:	0.015901	0.010143	0.013712	0.012593	0.017051	0.014939	0.014145	0.025557	0.015806	0.00867	0.02191	0.013681
std:	0.126099	0.100713	0.117097	0.112219	0.130578	0.122223	0.118934	0.159866	0.12572	0.093112	0.148019	0.116967
CV:	*0.226774*	*0.195599*	*0.197688*	*0.29414*	*0.224477*	*0.185216*	*0.248043*	*0.412701*	*0.233604*	*0.168458*	*0.32936*	*0.196972*

Table 6.3 Corresponding Best Hamming Distance Threshold Value for each 252 Experimental Scenarios

	16kB/sec				32kB/sec				64kB/sec			
FE %	2 Nodes	3 Nodes	4 Nodes	5 Nodes	2 Nodes	3 Nodes	4 Nodes	5 Nodes	2 Nodes	3 Nodes	4 Nodes	5 Nodes
0	1	1	1	1	1	1	1	1	1	1	1	1
5	13	14	16	15	13	14	15	14	14	14	15	15
10	14	14	17	15	15	15	15	15	14	15	15	15
15	15	15	17	15	16	15	16	15	14	15	14	15
20	15	16	17	15	16	15	16	16	15	15	16	15
25	15	16	18	15	15	15	16	15	15	17	15	15
30	15	17	16	15	16	15	16	15	14	15	16	15
35	15	16	17	16	17	15	16	15	15	16	16	15
40	16	16	17	16	15	16	16	16	16	16	19	15
45	15	16	17	17	16	16	16	15	16	16	16	16
50	17	16	17	18	15	15	16	16	16	17	16	15
55	17	16	17	15	17	17	16	18	16	17	18	17
60	18	17	17	17	16	17	16	18	16	16	16	17
65	16	16	17	16	16	15	17	18	16	15	16	17
70	17	15	17	17	16	16	16	16	17	15	17	17
75	15	16	17	17	15	16	17	16	16	15	16	17
80	16	16	17	19	16	16	16	15	15	17	17	17
85	17	17	17	19	16	15	17	19	15	16	19	17
90	15	16	17	17	15	16	16	17	17	16	17	17
95	16	17	17	16	17	17	18	17	16	17	17	16
100	17	16	17	19	16	17	18	16	18	17	18	17
Mean:	*15.7*	*15.9*	*16.95*	*16.45*	*15.7*	*15.65*	*16.25*	*16.1*	*15.55*	*15.85*	*16.45*	*16*
Mode:	**15**	*16*	*17*	**15**	*16*	**15**	*16*	**15**	*16*	**15**	*16*	**15**
Median:	*15.5*	*16*	*17*	*16*	*16*	*15.5*	*16*	*16*	*16*	*16*	*16*	*16*
Variance:	1.484211	0.726316	0.155263	2.05	0.852632	0.765789	0.618421	1.778947	1.207895	0.871053	1.734211	0.947368
std:	1.218282	0.852242	0.394034	1.431782	0.923381	0.875094	0.786398	1.333772	1.099043	0.933302	1.316894	0.973329
CV:	*0.077598*	*0.0536*	*0.023247*	*0.087038*	*0.058814*	*0.055917*	*0.048394*	*0.082843*	*0.070678*	*0.058883*	*0.080054*	*0.060833*

From the resulting F-Scores in Table 6.2, it can be seen that there is a significant decrease in the mean F-Score value from scenario with 2 nodes to 5 nodes. This is consistent in all three transmission rates. The other significant observation that is constant across each scenario for 2 nodes to 5 nodes and all three transmission, is the decrease in the F-Score value from FE= 5% to 100%. Both these observations confirm that F-Score is affected by the number of erroneous frames present. As the number of nodes increased,

the number of frames would go up which results in the amount of possible erroneous frames to also to rise. The fascinating thing about these results is that even at the unrealistic FE = 100%, the F-Score value still stayed approximately around 0.30 to 0.65. This alludes to the fact that even though the amount of error does slightly affect the F-Score value it is not enough to make finding an effective Hamming distance impossible. This can be seen in the best threshold results in Table 6.3. Note that the first row of Table 6.3 was omitted from the calculation of the statistical measures as FE% = 0 would only yield the HD values of the side channel frames. The resulting best threshold from all the 252 scenarios is seen to be 15 as it is the most frequently seen value. This is determined by the mode values going across the table. The second most frequently seen threshold value is 16. This Hamming distance thresholds of 15 or 16 seems to appear frequently as the most effective. Seeing as they fall outside the 99% confidence interval of the Hamming population mean of non-side channel frames and below the upper bound mean of the side channel frames, it can determined to be the most effective threshold value for use in detection of side channel.

Chapter 7

Conclusion and Future Works

7.1 Summary

The concept of the side channel was seen to be a domain specific problem where the existing protocol could be exploited to allow for sensitive information to be passed without knowledge of its presence. It also exploits the decentralized infrastructure of MANETs and employed the use of different CRC polynomials to disguise frames as in error. This is what makes it so hard to detect when errors are prone in this type of environment. Groups of malicious nodes are able to communicate based on using an agreed upon CRC polynomial that is different from the other nodes in the network. The CRC32 polynomial presented and explored were the default and Koopman. The default CRC polynomial was designated as the normal communication while Koopman was used for side channel.

Some prior detection metrics were presented such as frame error rate (FER) and request to send and clear to send (RTS & CTS). Both these metrics presented some weakness that called for a new metric to be found. The Hamming distance metric was introduced as a possible detection measure for side channel communication. The concept of the delta CRC was also introduce as the motivation for the discovery of the Hamming distance metric. This delta CRC calculated the numeric difference between CRC values

generated by the default and Koopman polynomials to see if this yielded values that clustered into a unique area. The results show that no such cluster was found as the numerical CRC values were uniformly distributed. This sparked the idea to change the approach to finding the number of bits different between CRC values generated by two different CRC polynomial such as the default and Koopman. The hypothesis was that the Hamming distance value between two CRC values using two different CRC polynomials was greater than ones that were generated by the same polynomial but was naturally corrupted during transmission. The preliminary evaluation results showed that there was indeed a significant enough difference that warranted further investigation into the use of the Hamming distance as a metric for detection.

This led into the survey of finding a simulator that could supported the delivery of intentionally corrupted frames. The simulator that showed some promise was the QualNet Network Simulator. After some failed attempts to modify the simulator to include the concept of a Frame Check Sequence, it was decided to go with a different approach. This approach combines the real data capture with simulation and was introduced as the hybrid testing environment. The real portion consisted of using the AirPcap adapter along with Wireshark to capture real data frames for use in the MATLAB/Simulink simulator. The simulator was responsible for taking these frames and recalculating the CRC values based on the polynomial specified and simulated the channel properties to include the possibility for error. The final step of the simulator calculated the Hamming distance values between CRC values of frames that used the

default and the Koopman polynomials. The output data is then used for validation and evaluation of the Hamming distance metric.

Validation is done with the use of a well know classification algorithm called the Perceptron & Pocket Learning algorithm. The Perceptron Learning algorithm uses feature(s) to classify by finding an equation of a line that separates the data into different classes. The only limitation is that it can only deal with linearly separable data. If it is not linearly separable, then the algorithm will not converge and may run infinitely. In order to handle linearly non-separable data, the Pocket algorithm was developed to keep track of the line equation that was able to separate the data as best as it can. Like many classification algorithms, the feature(s) chosen is very important and can determine how effectively it can separate the data into different classes. If the algorithm performs well with certain feature(s), then it can be validated as a good feature to use distinguishing different classes. In this case the feature used to distinguish side channel frames for non-side channel frames is the Hamming distance metric. The results showed that the Hamming distance metric was a valid feature to use for the classification of side channel frames and non-side channel frames. However the effectiveness was seen to change based on the amount of error in the network. This led to the idea of measuring how effect the Hamming distance metric is for different network scenarios.

The evaluation of how effective the Hamming distance metric for a particular network scenario is done through the use of scores such as true positives, true negative, false positives, and false negatives. Other metrics such as precision, sensitivity/recall,

specificity and accuracy were also used. It was seen that varying the Hamming distance thresholds yielded different score results. Some of the scores seem to minimize or maximized at certain threshold values. A composite of precision and sensitivity/recall called the F-Score determined to be the best indicator of the most effective threshold to use for detection for that particular scenario. It was shown that an effective threshold of 15 was found to be the most effective for one particular 5 node scenario that was ran but there was no claim to prove that these results could be generalized to other types of scenarios. This started the investigation into finding a common or most effective threshold that could be generalized to other types of scenarios.

In order to pin point a general effective threshold to use for detection, there needs to be a battery of test cases representing all possible scenarios. The only way is to generate all frame combinations of size 2342 bits that could be used to find all possible Hamming distance values that could be generated by both the default CRC and Koopman polynomials. The average Hamming distance values between CRC bit strings that were generated by the default versus Koopman polynomial can be calculated and compared to the average Hamming distance values for naturally corrupted bits strings that use just the default polynomial. If the averages were far enough apart, then any Hamming distance threshold value between them would suffice for detection. As it would be infeasible to generate all 2^{2342} different frames, the confidence interval was calculated for both 95% and 99% to predict what these averages would be. The results showed that with 99% confident, an effective threshold would lie above 8 and below 17. In order to test this, a total of 252 scenarios were created which varied the number nodes, the transmission

rates and the percentage of frame error. The results showed that the most effective threshold was a Hamming distance value of 15 which was frequently chosen as the best across all 252 scenarios. This confirmed with the confidence interval results.

7.2 Evaluating the Weaknesses

The purpose of this section is to explore possible weaknesses that relate to using the Hamming distance as a metric for detection of side channel communication. It also hopes to spark further work in strengthening the approach or give new direction in finding a better more resilient metric.

The first weakness with this approach is that it can only be applied to a wireless network environment which assumes that the nodes are using the default CRC polynomial while the adversary nodes use the Koopman CRC polynomial. The results cannot be applied to other possible CRC polynomials that may exist. In other words if the adversary nodes use a different CRC polynomial other than Koopman then the Hamming distance metric might fail. If the adversary nodes use another technique to corrupt the CRC value found in the FCS field then this metric might also fail.

A second weakness is the lack of scalability in the experiments as the number of nodes is limited to the availability of physical equipment such as laptops or tablets. As it was seen, the number of nodes did have an effect on the amount of possible errors that could occur in the network which could weaken the effectiveness of the Hamming distance metric. A simulation solution would be able to confirm this. However the results

still show that this Hamming distance metric did hold up to high unreasonable or unrealistic percentages of frame error such as 90%-100%.

One last weakness relates to the channel model used in the simulator portion of the hybrid testing environment. Even though it was designed to use real SNR values sensed from the real data captured, it is still a model that might not represent the real world. However these models do a very good job. This weakness is common when it comes to working with simulation where the only way to confirm the results is being able to reproduce it in a real life scenario.

7.3 Future Works

Throughout the thesis there were many mentions to possible future works that could be done. The first that was mentioned was the idea to take this side channel problem and apply it to a wired network that was more centralized. The question to answer is if this type of centralized environment would help or hinder the concealment of the side channel. In regards to environment, the second mention was to investigate the effectiveness of the Hamming distance metric in a MANET. This requires further exploration in mobility patterns and parameters that could cause the Hamming distance metric to become ineffective in detection of side channel communication. A third future work that may help improve the effectiveness of the Hamming distance metric is investigating the window size for sampling. It was mentioned that if an ideal window

size was found that captured less legitimate errors then this would reduce the false negative and false positive scores. This could then be furthered by applying both the Hamming distance metric and a window sizing technique to develop a detection algorithm or system. A forth future work is to compare the effectiveness of using the FER metric versus the Hamming distance metric. Finally the last future work is exploring the idea of finding CRC polynomials that would be better disguise intentionally corrupting frames as legitimate error. The direction could be taking a second look at the delta CRC metric or doing an exhaustive search in the space of all possible 2^{2342} combination of frame bit strings.

References

[1] R. R. Roy, Handbook of Mobile Ad Hoc Networks for Mobility Models, New York: Springer, 2011.

[2] R. Amirtharajan and J. B. B. Rayappan, "Steganography - Time to Time: A Review," *Research Journal of Information Technology,* vol. 5, no. 2, pp. 53-66, 2013.

[3] M. Odor, B. Nasri, M. Salmanian, P. C. Mason, M. Vargas Martin and R. Liscano, "A Frame Handler Module for a Side-Channel in Mobile Ad Hoc Networks," in *Local Computer Networks Workshop on Security in Communications Networks (SICK)*, Zurich, Switzerlan, pp.930-936, 2009.

[4] S. S. Kumaar, M. Mangai, N. Fernando, J. V. Daniel and R. Prabhu, "A Survey of Various Attacks in Mobile Ad Hoc Networks," *International Journal of Computer Science and Mobile Computing,* vol. 2, no. 10, pp. 171-185, 2013.

[5] M. Abolhasan, T. Wysocki and E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks, Ad Hoc Networks," *Ad Hoc Networks,* vol. 2, no. 1, pp. 1-22, 2004.

[6] D. V. Sarwate, "Computation of cyclic redundancy checks via table look-up," *Communications of the ACM,* vol. 31, no. 8, pp. 1008-1013, 1988.

[7] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *International Conference on Dependable Systems and Networks (DSN)*, pp.145-154, 2004.

[8] T. Baicheva, S. Dodunekov and P. Kazakov, "Undetected error probability performance of cyclic redundancy-check codes of 16-bit redundancy," *IEE Proceedings - Communications,* vol. 147, no. 5, pp. 253-256, 2000.

[9] P. Koopman, "32-bit cyclic redundancy codes for Internet applications," *International Conference on Dependable Systems and Networks,* pp. 459-468, 2002.

[10] T. G. Handel and M. T. Standford, "Data hiding in the OSI network model," *Lecture Notes in Computer Science (LNCS),* vol. 1174, pp. 23-38, 1996.

[11] K. Szczypiorski, "HICCUPS: Hidden Communication System for Corrupted Networks," *International Multi-Conference on Advanced Computer Systems (ACS),* pp.31-40, 2003.

[12] B. Jankowski, W. Mazurczyk and K. Szczypiorski, "Information hiding using improper frame padding," *International Telecommunications Network Strategy and Planning Symposium (NETWORKS),* pp.27-30, 2010.

[13] Y. Zhang, W. Lee and Y.-A. Huang, "Intrusion detection techniques for mobile wireless networks," *ACM Wireless Networks,* vol. 9, no. 5, pp. 545-556, 2003.

[14] T. Hayajneh, P. Krishnamurthy, D. Tipper and T. Kim, "Detecting malicious packet dropping in the presence of collisions and channel errors in wireless ad hoc networks," *International Conference on Communications,* pp.1062-1067, 2009.

[15] K. Xu, M. Gerla and S. Bae, "How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks?," *Global Telecommunications Conference (GLOBCOM),* vol. 1, pp. 72-76, 2002.

[16] D. E. Denning, "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering,* vol. 13, no. 2, pp. 222-232, 2006.

[17] M. Li, M. Salmanian, P. C. Mason, V. Chea, M. Vargas Martin and R. Liscano, "A realistic implementation for simulating side-channel in mobile ad hoc networks," *Proceedings of the Military Modeling & Simulation Symposium (MMS),* pp.2-10, 2013.

[18] N. Madtha, M. Vargas Martin, R. Liscano, B. Moore, M. Salmanian, M. Li and P. C. Mason, "Detection of side-channel communication in ad hoc networks using request to send (RTS) messages," *Canadian Conference on Electrical and Computer Engineering (CCECE),* pp.1-6, 2014.

[19] "Simulink - Simulation and Model-Based Design," MathWorks - Matlab, 1994-2015. [Online]. Available: http://www.mathworks.com/products/simulink/. [Accessed April 2014].

[20] K. H. Rosen, "Coding Theory," in *Applications of Discrete Mathematics*, New York, McGraw-Hill Higher Education, 2007, pp. 73-95.

[21] A. Najafizadeh, R. Liscano, M. Vargas Martin, P. Mason and M. Salmanian, "Challenges in the Implementation and Simulation for Wireless Side-Channel based on Intentionally Corrupted FCS," in *International Conference on Ambient Systems, Networks and Technologies (ANT)*, pp.165-172, 2011.

[22] "The Network Simulator - NS2," NS2, 1995-2011. [Online]. Available: http://www.isi.edu/nsnam/ns/. [Accessed April 2014].

[23] "QualNet | Scalable Networks Technologies," Scalable Networks Technologies, 2008-2014. [Online]. Available: http://web.scalable-networks.com/content/qualnet. [Accessed August 2014].

[24] "SteelCentral Network Performance Management | Riverbed AirPcap | Riverbed," Riverbed Technology, 2002-2015. [Online]. Available: http://www.riverbed.com/products/performance-management-control/network-performance-management/wireless-packet-capture.html. [Accessed April 2014].

[25] "Wireshark - About," Wireshark, 1998-2014. [Online]. Available: https://www.wireshark.org/about.html. [Accessed April 2014].

[26] "CaptureSetup/WLAN - The Wireshark Wiki," Wireshark, 1998-2014. [Online]. Available: http://wiki.wireshark.org/CaptureSetup/WLAN. [Accessed April 2014].

[27] R. Rojaz, "Perceptron Learning," in *Neural Networks: A Systematic Introduction*, Berlin, Springer Science & Business Media, 1996, pp. 77-99.

[28] M. Sokolova, N. Japkowicz and S. Szpakowicz, "Beyond accuracy, F-score and ROC : a Family of Discriminant Measures for Performance Evaluation," *Lecture Notes in Computer Science (LNCS),* vol. 4304, pp. 1015-1021, 2006.

[29] "Maple 18 - Technical Computing Software for Engineers, Mathematicians, Scientists, Instructors and Students - Maplesoft," Maplesoft, a Division of Waterloo Maple Inc., 1980-2015. [Online]. Available: http://www.maplesoft.com/products/maple/. [Accessed January 2015].

[30] A. Field, Discovering Statistics Using SPSS, Thousand Oaks, California: SAGE Publications Ltd., 2009.

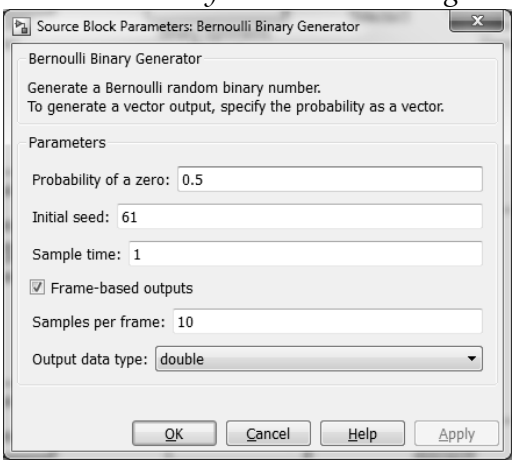
[31] G. W. Snedecor and W. G. Cochran, Statistical Methods, Ames: Iowa State University Press., 1989.

Table of Appendices

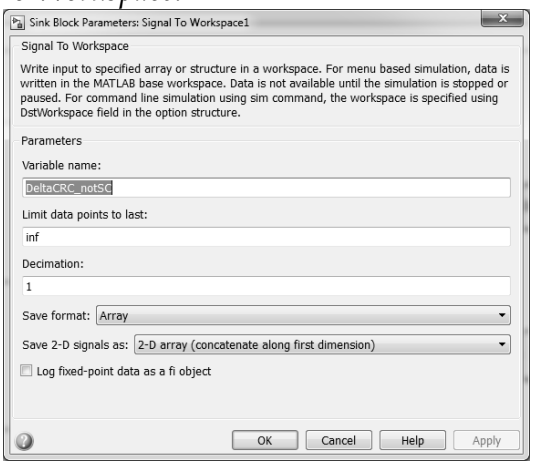
Appendix A

Configuration settings for the Simulink Modules in Figure 3.1:

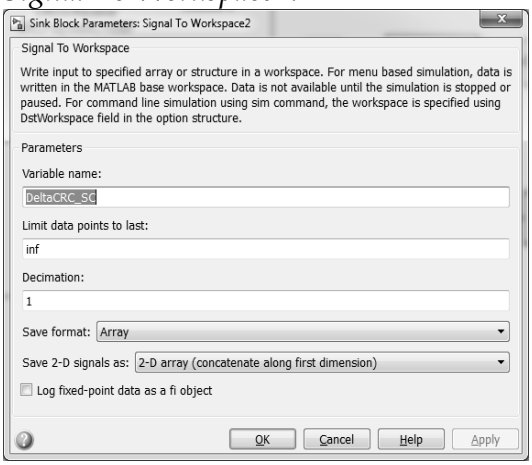
Bernoulli Binary Generator Settings:



Signal To Workspace:

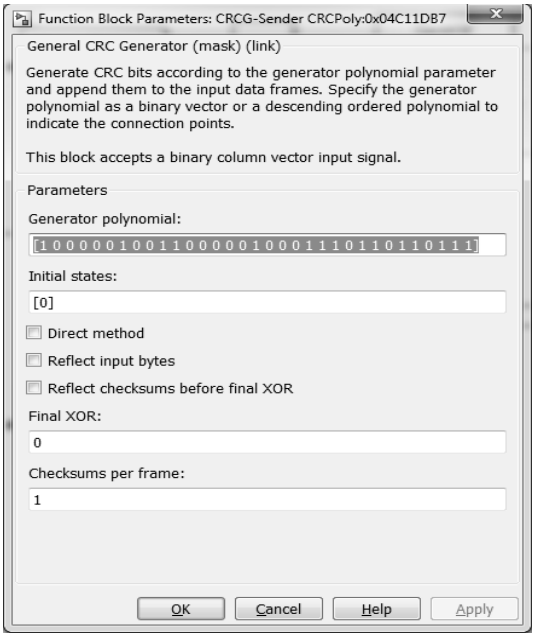


Signal To Workspace2:

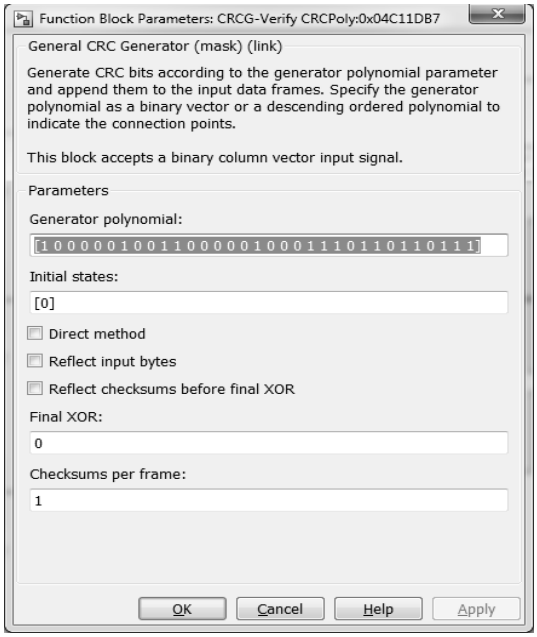


Section 1 – General CRC Generators

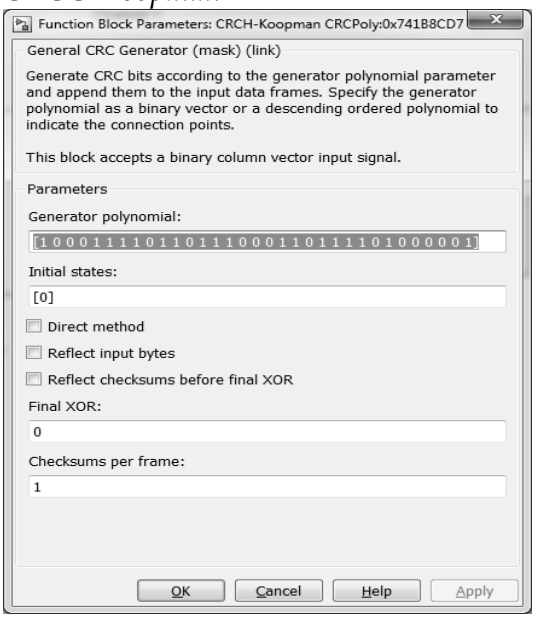
CRCG-Sender:



CRCG-Verify:

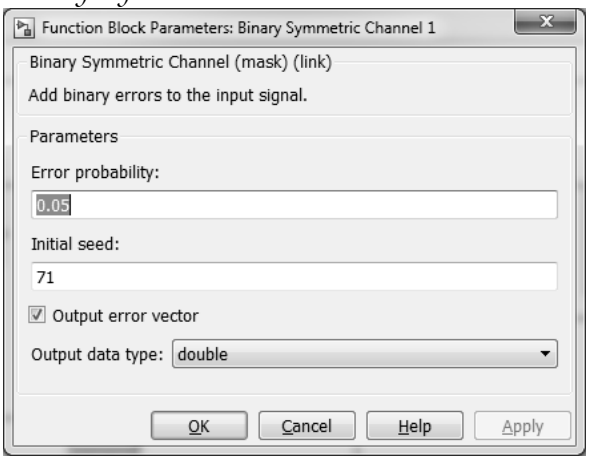


CRCG-Koopman

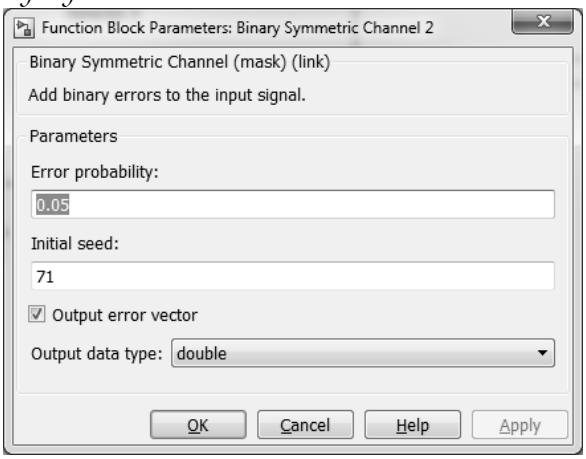


Section 2 – Binary Symmetric Channels

Binary Symmetric Channel 1:

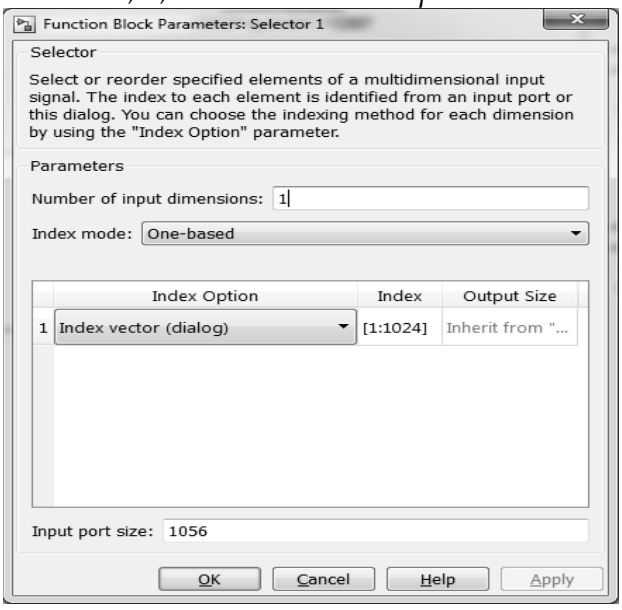


Binary Symmetric Channel 2:



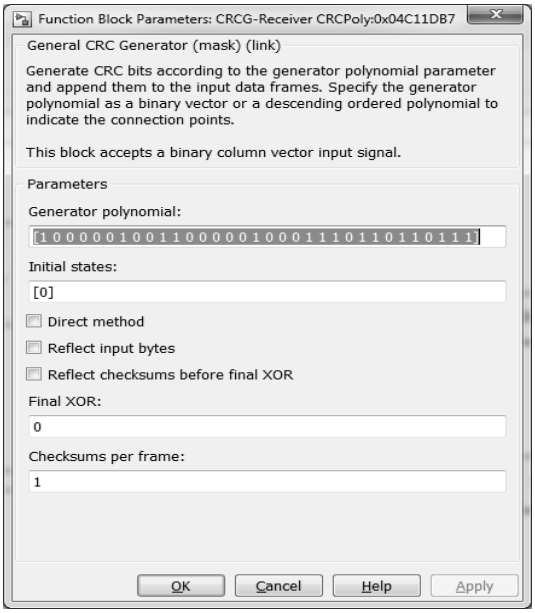
Section 3 – Selectors

Selector 1, 2, & 3 have the same parameters:

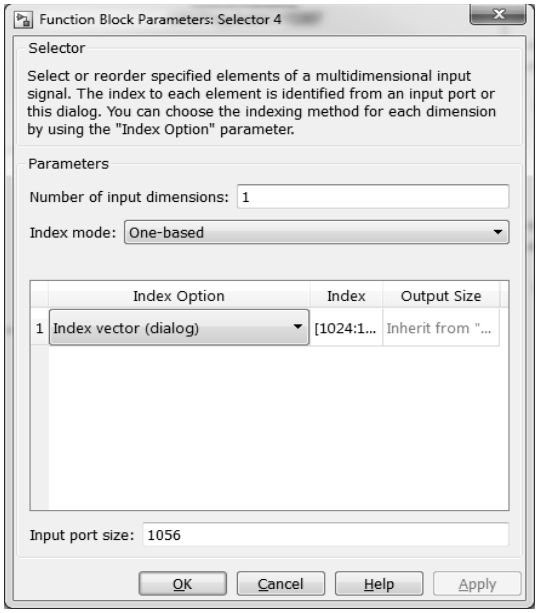


Section 4 – Receiver CRC Generator

CRCG-Receiver:

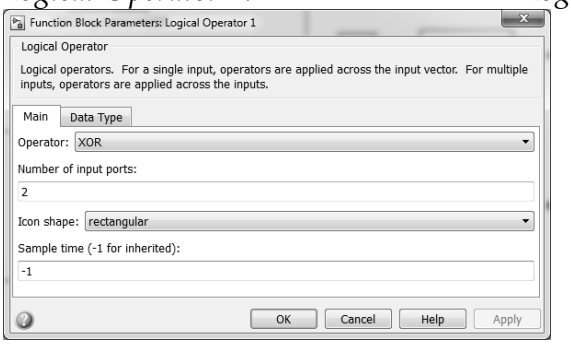


Selector 4:

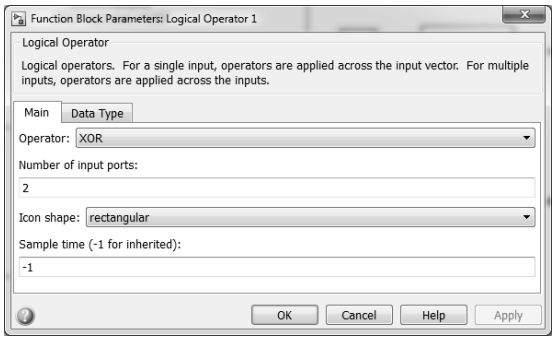


Section 5 – Delta CRC calculators

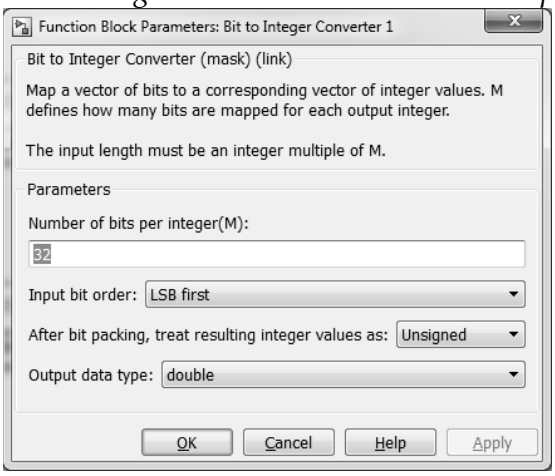
Logical Operator 1:



Logical Operator 2:



Bit to Integer Converter 1 & 2 have same parameters:



MATLAB Script to Test Preliminary Hamming Distance Metric:

```
t=fix(clock);
year=num2str(t(1));
month=num2str(t(2));
day=num2str(t(3));
hour=num2str(t(4));
min=num2str(t(5));
sec=num2str(t(6));

execTime=strcat(year,month,day,hour,min,sec);

%Default
DNEvsDEHDresults=[]; %stores the HD results after comparision
DNEvsDEMean=0; %stores the mean
DNEvsDEMedian=0; %stores the median
DNEvsDEMode=0; %srotes mode
DNEvsDEVariance=0; %stores the variancesum[ (xi -mean)2 ]/n
DNEvsDESTDev=0; %sqrt of variance

%Koopman woError
DNEvsKNEHDresults=[]; %stores the HD results after comparision
DNEvsKNEMean=0; %stores the mean
DNEvsKNEMedian=0; %stores the median
DNEvsKNEMode=0; %srotes mode
DNEvsKNEVariance=0; %stores the variancesum[ (xi -mean)2 ]/n
DNEvsKNESTDev=0; %sqrt of variance

%Koopman wiError
DNEvsKEHDresults=[]; %stores the HD results after comparision
DNEvsKEMean=0; %stores the mean
DNEvsKEMedian=0; %stores the median
DNEvsKEMode=0; %srotes mode
DNEvsKEVariance=0; %stores the variancesum[ (xi -mean)2 ]/n
DNEvsKESTDev=0; %sqrt of variance

%Castagnoli woError
DNEvsCNEHDresults=[]; %stores the HD results after comparision
DNEvsCNEMean=0; %stores the mean
DNEvsCNEMedian=0; %stores the median
DNEvsCNEMode=0; %srotes mode
DNEvsCNEVariance=0; %stores the variancesum[ (xi -mean)2 ]/n
DNEvsCNESTDev=0; %sqrt of variance

%Castagnoli wiError
DNEvsCEHDresults=[]; %stores the HD results after comparision
DNEvsCEMean=0; %stores the mean
DNEvsCEMedian=0; %stores the median
DNEvsCEMode=0; %srotes mode
DNEvsCEVariance=0; %stores the variancesum[ (xi -mean)2 ]/n
DNEvsCESTDev=0; %sqrt of variance

frCC=1;
blockCC=1;
while (blockCC < length(DNE))
    %need to initialize all string CRC bit values with the first value in first index of that block
    crcDefNoEr=num2str(DNE(blockCC));
    crcDefEr=num2str(DE(blockCC));
```

```
    crcKoopNoEr=num2str(KNE(blockCC));
    crcKoopEr=num2str(KE(blockCC));
    crcCastNoEr=num2str(CNE(blockCC));
    crcCastEr=num2str(CE(blockCC));

    %initialize all comparison values
    DNEvsDE=0;
    DNEvsKNE=0;
    DNEvsKE=0;
    DNEvsCNE=0;
    DNEvsCE=0;


    for n = 0:31 %increments from the first bit to the last 32 bit block
        arrayIn=blockCC + n; %the array index is made up of getting block added with the bit of the for
loop

        bitValDNE=DNE(arrayIn);
        bitValDE=DE(arrayIn);
        bitValKNE=KNE(arrayIn);
        bitValKE=KE(arrayIn);
        bitValCNE=CNE(arrayIn);
        bitValCE=CE(arrayIn);

        %perform Comparisons by XOR then add to find HD
        DNEvsDE=DNEvsDE+(bitxor(bitValDNE,bitValDE));
        DNEvsKNE=DNEvsKNE+(bitxor(bitValDNE,bitValKNE));
        DNEvsKE=DNEvsKE+(bitxor(bitValDNE,bitValKE));
        DNEvsCNE=DNEvsCNE+(bitxor(bitValDNE,bitValCNE));
        DNEvsCE=DNEvsCE+(bitxor(bitValDNE,bitValCE));

    end

    DNEvsDEHDresults(frCC)=DNEvsDE; %stores the HD results after comparision
    DNEvsKNEHDresults(frCC)=DNEvsKNE; %stores the HD results after comparision
    DNEvsKEHDresults(frCC)=DNEvsKE; %stores the HD results after comparision
    DNEvsCNEHDresults(frCC)=DNEvsCNE; %stores the HD results after comparision
    DNEvsCEHDresults(frCC)=DNEvsCE; %stores the HD results after comparision

    frCC=frCC+1;
    blockCC=blockCC+32; %increments to the next block of CRC bits
end

%for testing
%DNEvsDEHDresults
%DNEvsKNEHDresults
%DNEvsKEHDresults

DNEvsDEMean=mean(DNEvsDEHDresults); %stores the mean
DNEvsKNEMean=mean(DNEvsKNEHDresults); %stores the mean
DNEvsKEMean=mean(DNEvsKEHDresults); %stores the mean
DNEvsCNEMean=mean(DNEvsCNEHDresults); %stores the mean
DNEvsCEMean=mean(DNEvsCEHDresults); %stores the mean

DNEvsDEMedian=median(DNEvsDEHDresults); %stores the median
DNEvsKNEMedian=median(DNEvsKNEHDresults); %stores the median
DNEvsKEMedian=median(DNEvsKEHDresults); %stores the median
DNEvsCNEMedian=median(DNEvsCNEHDresults); %stores the median
DNEvsCEMedian=median(DNEvsCEHDresults); %stores the median
```

```
DNEvsDEMode=mode(DNEvsDEHDresults); %srotes mode
DNEvsKNEMode=mode(DNEvsKNEHDresults); %srotes mode
DNEvsKEMode=mode(DNEvsKEHDresults); %srotes mode
DNEvsCNEMode=mode(DNEvsCNEHDresults); %srotes mode
DNEvsCEMode=mode(DNEvsCEHDresults); %srotes mode

DNEvsDEVariance=var(DNEvsDEHDresults); %stores the variancesum[ (xi -mean)2 ]/n
DNEvsKNEVariance=var(DNEvsKNEHDresults); %stores the variancesum[ (xi -mean)2 ]/n
DNEvsKEVariance=var(DNEvsKEHDresults); %stores the variancesum[ (xi -mean)2 ]/n
DNEvsCNEVariance=var(DNEvsCNEHDresults); %stores the variancesum[ (xi -mean)2 ]/n
DNEvsCEVariance=var(DNEvsCEHDresults); %stores the variancesum[ (xi -mean)2 ]/n

DNEvsDESTDev=std(DNEvsDEHDresults); %sqrt of variance
DNEvsKNESTDev=std(DNEvsKNEHDresults); %sqrt of variance
DNEvsKESTDev=std(DNEvsKEHDresults); %sqrt of variance
DNEvsCNESTDev=std(DNEvsCNEHDresults); %sqrt of variance
DNEvsCESTDev=std(DNEvsCEHDresults); %sqrt of variance

strFName = strcat('HDStats_NumOfFrames',num2str(frCC-1),'_wiAWGNSNR10','_',execTime);
strFNameOut=strcat(strFName,'.txt');
fOut = fopen(strFNameOut,'wt');

fprintf(fOut,'HD Comparision: Default woErrors vs Default wiError \n');
fprintf(fOut,'Mean: %-4.3f \n',DNEvsDEMean);
fprintf(fOut,'Median: %-4.3f \n',DNEvsDEMedian);
fprintf(fOut,'Mode: %-4.3f \n',DNEvsDEMode);
fprintf(fOut,'Variance:%-4.3f \n',DNEvsDEVariance);
fprintf(fOut,'Standard Deviation: %-4.3f \n',DNEvsDESTDev);
fprintf(fOut,'Coefficient of Variation (CV): %-4.3f \n',(DNEvsDESTDev/DNEvsDEMean));
fprintf(fOut,'\n');

fprintf(fOut,'HD Comparision: Default woErrors vs Koopman woError \n');
fprintf(fOut,'Mean:%-4.3f \n',DNEvsKNEMean);
fprintf(fOut,'Median:%-4.3f \n',DNEvsKNEMedian);
fprintf(fOut,'Mode:%-4.3f \n',DNEvsKNEMode);
fprintf(fOut,'Variance:%-4.3f \n',DNEvsKNEVariance);
fprintf(fOut,'Standard Deviation:%-4.3f \n',DNEvsKNESTDev);
fprintf(fOut,'Coefficient of Variation (CV): %-4.3f \n',(DNEvsKNESTDev/DNEvsKNEMean));
fprintf(fOut,'\n');

fprintf(fOut,'HD Comparision: Default woErrors vs Koopman wiError \n');
fprintf(fOut,'Mean:%-4.3f \n',DNEvsKEMean);
fprintf(fOut,'Median:%-4.3f \n',DNEvsKEMedian);
fprintf(fOut,'Mode:%-4.3f \n',DNEvsKEMode);
fprintf(fOut,'Variance:%-4.3f \n',DNEvsKEVariance);
fprintf(fOut,'Standard Deviation:%-4.3f \n',DNEvsKESTDev);
fprintf(fOut,'Coefficient of Variation (CV): %-4.3f \n',(DNEvsKESTDev/DNEvsKEMean));
fprintf(fOut,'\n');

fprintf(fOut,'HD Comparision: Default woErrors vs Castagnoli woError \n');
fprintf(fOut,'Mean:%-4.3f \n',DNEvsCNEMean);
fprintf(fOut,'Median:%-4.3f \n',DNEvsCNEMedian);
fprintf(fOut,'Mode:%-4.3f \n',DNEvsCNEMode);
fprintf(fOut,'Variance:%-4.3f \n',DNEvsCNEVariance);
fprintf(fOut,'Standard Deviation:%-4.3f \n',DNEvsCNESTDev);
fprintf(fOut,'Coefficient of Variation (CV): %-4.3f \n',(DNEvsCNESTDev/DNEvsCNEMean));
fprintf(fOut,'\n');

fprintf(fOut,'HD Comparision: Default woErrors vs Castagnoli wiError \n');
fprintf(fOut,'Mean:%-4.3f \n',DNEvsCEMean);
```

```
fprintf(fOut,'Median:%-4.3f \n',DNEvsCEMedian);
fprintf(fOut,'Mode:%-4.3f \n',DNEvsCEMode);
fprintf(fOut,'Variance:%-4.3f \n',DNEvsCEVariance);
fprintf(fOut,'Standard Deviation:%-4.3f \n',DNEvsCESTDev);
fprintf(fOut,'Coefficient of Variation (CV): %-4.3f \n',(DNEvsCESTDev/DNEvsCEMean));
fprintf(fOut,'\n');

fclose(fOut);
```

Appendix B

Java Program for Reformat Wireshark Frame Data Code:

```
import java.lang.*;
import java.text.DecimalFormat;
import java.io.*;
import java.util.*;

public class tSharkParser{
   public static void main(String[] args){ //at run line input format: FilenameToParse
TotalNumberOfPackets Matlab[-m] type[-bin] space[-s] keepFCS[-fcs]
      char[] decimForm = {'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'};
        String[] binaryForm =
{"0000","0001","0010","0011","0100","0101","0110","0111","1000","1001","1010","1011","1100","1101","1110",
"1111"};
      String[] binaryFormS = {"0 0 0 0","0 0 0 1","0 0 1 0","0 0 1 1","0 1 0 0","0 1 0 1","0 1 1 0","0 1
1 1","1 0 0 0","1 0 0 1","1 0 1 0","1 0 1 1","1 1 0 0","1 1 0 1","1 1 1 0","1 1 1 1"};

      String fNameTrans= args[0];
      int totNumPack=Integer.parseInt(args[1]);
      String isMatlab="-n";
      String hexVSbin="-hex";
      String isSpaced="-n";
      String keepFCS="-n";
      if(args.length==3){
         isMatlab = args[2];
      }
      if(args.length==4){
         hexVSbin = args[3];
      }
      if(args.length==5){
         isSpaced = args[4];
      }
      if(args.length==6){
         keepFCS = args[5];
      }

      String[] binSeqData = new String[totNumPack];
      String[] hexSeqData = new String[totNumPack];
        String[] recFCS = new String[totNumPack];
        double[] decFCS = new double[totNumPack];

      Queue<String> theFCS = new LinkedList<String>();

      if(isMatlab.contains("-m")){
         //System.out.println("File: "+ fNameTrans+" Display type: Matlab input formatted");
      }else{
         System.out.println("File: "+ fNameTrans+" Display type: "+ hexVSbin + " " + isSpaced + " " +
keepFCS);
      }
      try {
         String line = "";
         String nextLine =""; //stores the last line before we hit the new packet
         String binSeqDataTemp =""; //temp store our data value, concatinate each binary
```

```
          String recFCSTemp =""; //temp store for the FCS, concatinate each byte pair
            String hexSeqDataTemp="";
          File inFile = new File(fNameTrans);
          BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(inFile)));
          int index = 0;
          int flagFirstRun = 0;
          /*
          *Read each line, split using spaces " "
          *The first entry of every line is 2 bytes
          *the first line, first entry is identified by 0000
          *the entry of each line contains period characters '.', check for those to signal last entry
          *everything in the middle is valid data in sets of 1 byte(two values) each seperated by space
          *the last 4 bytes(4 sets of two vlaues) is the fcs, treat that differently
          *still need to find a way to store the last 4
          */
          while ((nextLine = br.readLine())!= null && index < totNumPack){ //the nextLine read will fail,
so after the while must process the last line with the FCS
              if(flagFirstRun == 0){
                 line = nextLine;
                 nextLine = br.readLine();
                 flagFirstRun=1;
              }
              String[] tempLine = line.split(" ");
              String[] tempNextLine = nextLine.split(" ");
                 if(tempNextLine[0].length()==4){//checks to make sure it is not a newline and signals a
new packet
                 //process line
                 //System.out.println(line); //test print print each line


                 for(int i=1; i<tempLine.length; i++){
                    //do binary conversion for each byte(value pair) to 8bits
                    //also add each byte to the queue
                         if(tempLine[i].length()==2 && tempLine[i].indexOf(".")<0){
                             hexSeqDataTemp=hexSeqDataTemp+" "+tempLine[i];
                             char[] valAt = new char[2];
                             valAt[0] = tempLine[i].charAt(0);
                             valAt[1] = tempLine[i].charAt(1);
                             for(int f1=0; f1<2; f1++){
                                 int found=-1;
                                 for(int f2=0; f2<decimForm.length; f2++){
                                     if(valAt[f1]==decimForm[f2]){
                                         found = f2;
                                     }
                                 }

                                 if(found>-1){
                              if(isSpaced.contains("-s")){
                                 binSeqDataTemp=binSeqDataTemp +" "+binaryFormS[found];
                              }else{
                                 binSeqDataTemp=binSeqDataTemp +""+binaryForm[found];
                              }
                                 }
                             }

                             if(theFCS.size()<4){
                                 theFCS.add(tempLine[i]);
                             }else{//the queue is max out at 4
                                 //remove one and add one
                                 theFCS.remove();
```

```
                theFCS.add(tempLine[i]);
            }
        }

   }
   //after done process the line, line = nextLine
   line = nextLine;
}else{
   //process the last line with the fcs before going to new line
   for(int i=1; i<tempLine.length; i++){
      //do binary conversion for each byte(value pair) to 8bits
      //also add each byte to the queue
      if(tempLine[i].length()==2 && tempLine[i].indexOf(".")<0){
              hexSeqDataTemp=hexSeqDataTemp+" "+tempLine[i];
              char[] valAt = new char[2];
              valAt[0] = tempLine[i].charAt(0);
              valAt[1] = tempLine[i].charAt(1);
              for(int f1=0; f1<2; f1++){
                  int found=-1;
                  for(int f2=0; f2<decimForm.length; f2++){
                      if(valAt[f1]==decimForm[f2]){
                          found = f2;
                      }
                  }
                  if(found>-1){
                      if(isSpaced.contains("-s")){
                     binSeqDataTemp=binSeqDataTemp +" "+binaryFormS[found];
                   }else{
                      binSeqDataTemp=binSeqDataTemp +""+binaryForm[found];
                   }
                      }
              }
              if(theFCS.size()<4){
                  theFCS.add(tempLine[i]);
              }else{//the queue is max out at 4
                  //remove one and add one
                  theFCS.remove();
                  theFCS.add(tempLine[i]);
              }
          }

   }
   //System.out.println(line); //test print print each line
   for(int y=0; y<4; y++){
          recFCSTemp = theFCS.remove()+""+recFCSTemp;
       }

       double decFCSTemp=0.0;

       for(int fc1=7; fc1>-1; fc1--){
           int foundc=-1;
           for(int fc2=0; fc2<decimForm.length; fc2++){
               if(recFCSTemp.charAt(fc1)==decimForm[fc2]){
                   foundc = fc2;
               }
           }
      //System.out.println(foundc+" "+ foundc*(Math.pow(16,(7-fc1)))); //testing
           if(foundc>-1){
               decFCSTemp=decFCSTemp+(foundc*(Math.pow(16,(7-fc1))));
           }
```

```
                    }
                String binSeqDataTempminusFCS="";
                String hexSeqDataTempminusFCS="";
                if(keepFCS.contains("-fcs") && !(isMatlab.contains("-m"))){
                   binSeqDataTempminusFCS = binSeqDataTemp;
                   hexSeqDataTempminusFCS = hexSeqDataTemp;
                }else{
                   if(isSpaced.contains("-s")){
                      binSeqDataTempminusFCS = binSeqDataTemp.substring(0,binSeqDataTemp.length()-
(32*2));
                      hexSeqDataTempminusFCS = hexSeqDataTemp.substring(0,hexSeqDataTemp.length()-
(12*2));
                   }else{
                      binSeqDataTempminusFCS = binSeqDataTemp.substring(0,binSeqDataTemp.length()-32);
                      hexSeqDataTempminusFCS = hexSeqDataTemp.substring(0,hexSeqDataTemp.length()-12);
                   }
                    }
                if(hexVSbin.contains("-hex") && !(isMatlab.contains("-m"))){
                   System.out.println(index+1+" "+hexSeqDataTempminusFCS+" "+recFCSTemp+" "+ decFCSTemp);
//test print print each line
                }else{
                   if(isMatlab.contains("-m")){
                      System.out.println(binSeqDataTempminusFCS); //print out for Matlab input
                   }else{
                      System.out.println(index+1+" "+binSeqDataTempminusFCS+" "+recFCSTemp+" "+
decFCSTemp); //test print print each line
                   }
                }
                hexSeqData[index] = hexSeqDataTempminusFCS;
                    binSeqData[index] = binSeqDataTempminusFCS;
                recFCS[index] = recFCSTemp;
                    decFCS[index] = decFCSTemp;
                //after line is process line readIn
                index++;
                binSeqDataTemp="";
                recFCSTemp="";
                    hexSeqDataTemp="";
                    decFCSTemp=0.0;
                line= br.readLine();
             }
          }
          //the nextLine read will fail, after must process the last line with the FCS
          br.close();


   }
}
```

Appendix C

Java Perceptron & Pocket Learning Algorithm Code:

```
/**
 *The Pocket Algorithm
 *Visal Chea
 *
 *Perceptron Learning Algorithm Code adapted from:
 *https://github.com/RichardKnop/ansi-c-perceptron
 *and
 *Dr Noureddin Sadawi http://people.brunel.ac.uk/~csstnns/tutorials.html
*/

import java.text.*;
import java.math.*;
import java.lang.*;
import java.io.*;
import java.util.*;

class PocketAlgorithmSCData
{
   static int MAX_ITER = 1000;
   static double LEARNING_RATE = 0.1;
   static int theta = 0;


   public static void main(String[] args){ //cmdLine format: FilenameToParse TotalNumberOfPackets
TestFileName TotNumberofTest isScatter

      String fNameTrans= args[0];
      int totNumPack=Integer.parseInt(args[1]);
      String fNameTest=args[2];
      int totNumTest=Integer.parseInt(args[3]);
      int isScatter = Integer.parseInt(args[4]); //0-scatter(x=y) 1-frameVSHD(x=frame#, y=HDvalue)
      //1 features = 1 variable x but plotted (x,x)
      int[] x = new int [totNumPack];
      int[] y = new int [totNumPack];
      int[] outputs = new int [totNumPack];

      //class 0 non SC
      int[] datax1=new int[totNumPack];
      int[] datay1=new int[totNumPack];
      int sizeData1=0;

      //class 1 SC
      int[] datax2=new int[totNumPack];
      int[] datay2=new int[totNumPack];
      int sizeData2=0;

      //testing points
      int[] datax3=new int[totNumTest];
      int[] datay3=new int[totNumTest];
      int sizeData3=0;
```

```
      double correctClassify=0;

      double correctSCClassify=0;
      double numOfSCinTest=0;


      //while loop to read in file with HD class(0-nonSC,1-SC[ftp])
      try {
         String line = "";
         File inFile = new File(fNameTrans);
         BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(inFile)));
         int index = 0;
         /*
         *Read each line, split using spaces " "
         * x y class
         *frame HD class(0-nonSC,1-SC[ftp])
         */
         while ((line = br.readLine())!= null && index < totNumPack){ //the nextLine read will fail, so
after the while must process the last line with the FCS
            //System.out.println("index: "+ index +"\n");
            String[] tempLine = line.split(" ");
            outputs[index] = Integer.parseInt(tempLine[1]);
            if(isScatter==0){//0-scatter(x=y)
               x[index]=Integer.parseInt(tempLine[0]);
               y[index]=Integer.parseInt(tempLine[0]);
               if(outputs[index]==0){ //in class 0
                  datax1[sizeData1]=x[index];
                  datay1[sizeData1]=y[index];
                  sizeData1++;
               }else{ //in class 1
                  datax2[sizeData2]=x[index];
                  datay2[sizeData2]=y[index];
                  sizeData2++;
               }
            }else if(isScatter==1){//1-frameVSHD(x=frame#, y=HDvalue)
               x[index]=index;
               y[index]=Integer.parseInt(tempLine[0]);
               if(outputs[index]==0){ //in class 0
                  datax1[sizeData1]=x[index];
                  datay1[sizeData1]=y[index];
                  sizeData1++;
               }else{ //in class 1
                  datax2[sizeData2]=x[index];
                  datay2[sizeData2]=y[index];
                  sizeData2++;
               }
            }


            //System.out.println(x[index]+"\t"+outputs[index]);
            index++;
         }
         br.close();

      } catch (FileNotFoundException ex) {
         //
      } catch (IOException ex) {
         //
      }
```

```
      double[] weights = new double[3];// 2 for input variables and one for bias
      double[] pocketWeights = new double[3];
      double localError, globalError;
      int p, iteration, output;

      weights[0] = randomNumber(0,1);// w1
      weights[1] = randomNumber(0,1);// w2
      weights[2] = randomNumber(0,1);// // this is the bias

      iteration = 0;
      double RMSE = 0;
      double bRMSE = 1;
      do {
         iteration++;
         globalError = 0;
         //loop through all instances (complete one epoch)
         for (p = 0; p < totNumPack; p++) {
            // calculate predicted class
            output = calculateOutput(theta,weights, x[p], y[p]);
            // difference between predicted and actual class values
            localError = outputs[p] - output;
            //update weights and bias
            weights[0] += LEARNING_RATE * localError * x[p];
            weights[1] += LEARNING_RATE * localError * y[p];
            weights[2] += LEARNING_RATE * localError;
            //weights[3] += LEARNING_RATE * localError;
            //summation of squared error (error value for all instances)
            globalError += (localError*localError);
         }


         //System.out.println("Iteration "+iteration+" : error = "+globalError);

         /* Root Mean Squared Error */
         RMSE = Math.sqrt(globalError/totNumPack);
         //System.out.println("Iteration "+iteration+" : RMSE = "+RMSE);

         if(RMSE < bRMSE){ //we have a new best value and new weights to put in pocket
            pocketWeights[0]=weights[0];
            pocketWeights[1]=weights[1];
            pocketWeights[2]=weights[2];
            bRMSE = RMSE; //new best RMSE value
         }

      } while (globalError != 0 && iteration<=MAX_ITER);
      System.out.println("\nTotal Iteration "+iteration+" : RMSE = "+RMSE);

      if(RMSE>0 && iteration>MAX_ITER){
         System.out.println("Did not converge best RMSE = " +bRMSE);
         if(bRMSE <= RMSE){ //it did not converge and the error went back up, revert back to the weights
in the pocket
            weights[0]=pocketWeights[0];
            weights[1]=pocketWeights[1];
            weights[2]=pocketWeights[2];
            System.out.println("=======\nPocket Decision boundary equation:");
            System.out.println(weights[0] +"*x + "+weights[1]+"*y +  "+weights[2]+" = 0 \n");
         }
      }else{
```

```
         System.out.println("=======\nDecision boundary equation:");
         System.out.println(weights[0] +"*x + "+weights[1]+"*y +  "+weights[2]+" = 0 \n");
      }

      //Here we test a few new points

      try {
         String lineTest = "";
         File inFileTest = new File(fNameTest);
         BufferedReader brTest = new BufferedReader(new InputStreamReader(new
FileInputStream(inFileTest)));
         int indexTest = 0;
         /*
         *Read each line, split using spaces " "
         *HD class number class(0-nonSC,1-SC[ftp])
         */
         int xTest=0;
         int yTest=0;
         int ActualOutput=0;
         while ((lineTest = brTest.readLine())!= null && indexTest < totNumTest){ //the nextLine read
will fail, so after the while must process the last line with the FCS
            String[] tempLine = lineTest.split(" ");
            if(isScatter==0){//0-scatter(x=y)
               xTest= Integer.parseInt(tempLine[0]);
               yTest= Integer.parseInt(tempLine[0]);
               datax3[sizeData3]=xTest;
               datay3[sizeData3]=yTest;
               sizeData3++;
            }else if(isScatter==1){//1-frameVSHD(x=frame#, y=HDvalue)
               xTest= indexTest;
               yTest= Integer.parseInt(tempLine[0]);
               datax3[sizeData3]=xTest;
               datay3[sizeData3]=yTest;
               sizeData3++;
            }
            ActualOutput = Integer.parseInt(tempLine[1]);
            output = calculateOutput(theta,weights, xTest, yTest);
            if(ActualOutput==1){
               numOfSCinTest++;
               //System.out.println("frameHD = "+yTest+", ActualClass = SC");
            }else{
               //System.out.println("frameHD = "+yTest+", ActualClass = nonSC");
            }

            if (output==1 && ActualOutput==1){
               correctClassify++;
               correctSCClassify++;
               //System.out.println("Algorithm classified correctly as SC\n");
            }else if(output==0 && ActualOutput==0){
               correctClassify++;
               //System.out.println("Algorithm classified correctly as nonSC\n");
            }else if(output==0 && ActualOutput==1){
               //System.out.println("classified incorrectly as SC\n");
            }else if(output==1 && ActualOutput==0){
               //System.out.println("classified inCorrectly as nonSC\n");
            }
            indexTest++;
         }
         brTest.close();
         //System.out.println("IndexTest: "+ indexTest +"\n");
```

```
         //System.out.println("correctClassify: "+ correctClassify +"\n");
         double correctAvgClassify = (correctClassify/(indexTest+0.0))*100;
         double correctAvgSCClassify = (correctSCClassify/numOfSCinTest)*100;
         System.out.println("Percentage of Correct Classification: "+ correctAvgClassify +"\n");
         System.out.println("Percentage of Correct SC Classified: "+ correctAvgSCClassify +"\n");

      } catch (FileNotFoundException ex) {
         //
      } catch (IOException ex) {
         //
      }

      System.out.print("datax1 = [");
      for (int x1=0; x1<sizeData1; x1++){
         System.out.print(" "+ datax1[x1]);
      }
      System.out.print("]\n");
      System.out.print("datay1 = [");
      for (int y1=0; y1<sizeData1; y1++){
         System.out.print(" "+ datay1[y1]);
      }
      System.out.print("]\n");

      System.out.print("datax2 = [");
      for (int x2=0; x2<sizeData2; x2++){
         System.out.print(" "+ datax2[x2]);
      }
      System.out.print("]\n");
      System.out.print("datay2 = [");
      for (int y2=0; y2<sizeData2; y2++){
         System.out.print(" "+ datay2[y2]);
      }
      System.out.print("]\n");

      System.out.print("datax3 = [");
      for (int x3=0; x3<sizeData3; x3++){
         System.out.print(" "+ datax3[x3]);
      }
      System.out.print("]\n");
      System.out.print("datay3 = [");
      for (int y3=0; y3<sizeData3; y3++){
         System.out.print(" "+ datay3[y3]);
      }
      System.out.print("]\n");

   }//end main

/**
 * returns a random double value within a given range
 * @param min the minimum value of the required range (int)
 * @param max the maximum value of the required range (int)
 * @return a random double value between min and max
 */
public static double randomNumber(int min , int max) {
    DecimalFormat df = new DecimalFormat("#.####");
    double d = min + Math.random() * (max - min);
    String s = df.format(d);
    double x = Double.parseDouble(s);
    return x;
}
```

Appendix C

```
 /**
  * returns either 1 or 0 using a threshold function
  * theta is 0range
  * @param theta an integer value for the threshold
  * @param weights[] the array of weights
  * @param x the x input value
  * @param y the y input value
  * @param z the z input value
  * @return 1 or 0
  */
 static int calculateOutput(int theta, double weights[], int x, int y)
 {
    double sum = x * weights[0] + y * weights[1] + weights[2];
    return (sum >= theta) ? 1 : 0;
 }

}
```

Appendix D

Calculated Evaluation Metric Results for Scenario 5Node@64kB/s with 19.82% FE:

(Increased Side Channel Frames)

Threshold	True(+)	False(+)	True(-)	False(-)	Accuracy	Sensitivity	Specificity	Precision	F-Score
1	37	5164	21260	0	0.8048	1.0000	0.8046	0.0071	0.0141
2	37	5125	21299	0	0.8063	1.0000	0.8060	0.0072	0.0142
3	37	5026	21398	0	0.8101	1.0000	0.8098	0.0073	0.0145
4	37	4807	21617	0	0.8183	1.0000	0.8181	0.0076	0.0152
5	37	4389	22035	0	0.8341	1.0000	0.8339	0.0084	0.0166
6	37	3748	22676	0	0.8584	1.0000	0.8582	0.0098	0.0194
7	37	2933	23491	0	0.8892	1.0000	0.8890	0.0125	0.0246
8	37	2129	24295	0	0.9195	1.0000	0.9194	0.0171	0.0336
9	37	1431	24993	0	0.9459	1.0000	0.9458	0.0252	0.0492
10	37	810	25614	0	0.9694	1.0000	0.9693	0.0437	0.0837
11	33	404	26020	4	0.9846	0.8919	0.9847	0.0755	0.1392
12	31	185	26239	6	0.9928	0.8378	0.9930	0.1435	0.2451
13	29	85	26339	8	0.9965	0.7838	0.9968	0.2544	0.3841
14	22	37	26387	15	0.9980	0.5946	0.9986	0.3729	0.4583
15	**19**	**13**	**26411**	**18**	**0.9988**	**0.5135**	**0.9995**	**0.5938**	**0.5507**
16	15	6	26418	22	0.9989	0.4054	0.9998	0.7143	0.5172
17	10	1	26423	27	0.9989	0.2703	1.0000	0.9091	0.4167
18	8	0	26424	29	0.9989	0.2162	1.0000	1.0000	0.3556
19	6	0	26424	31	0.9988	0.1622	1.0000	1.0000	0.2791
20	3	0	26424	34	0.9987	0.0811	1.0000	1	0.15
21	1	0	26424	36	0.9986	0.0270	1.0000	1	0.0526

Appendix E

Statistical and F-Score Results for the 5Node Scenario with FTP data frames:

T-Test results for Scenario 5Node@16kB/s with 15.86% FE (FTP data frames from port 51434)

t-Test: Two-Sample Assuming Equal Variances		
	FTP-Control	*FTP-Data*
Mean	15.81081081	15.97427386
Variance	9.935435435	7.959495737
Observations	37	4820
Pooled Variance	7.974147401	
Hypothesized Mean Difference	0	
df	4855	
t Stat	-0.350766244	
P(T<=t) one-tail	0.362889471	
t Critical one-tail	1.645167543	
P(T<=t) two-tail	0.725778942	
t Critical two-tail	1.960452729	

Confidence Interval Results Scenario 5Node@16kB/s with 15.86% FE (FTP data frames from port 51434)

Confidence Level=95% (non-side channel)	
Confidence Level(95.0%)	0.075939027
Lower bound mean	7.876270387
Upper bound mean	8.028148441

Confidence Level=95% (side channel)	
Confidence Level(95.0%)	0.079666629
Lower bound mean	15.89460723
Upper bound mean	16.05394049

Confidence Level=99% (non-side channel)	
Confidence Level(99.0%)	0.099817547
Lower bound mean	7.852391867
Upper bound mean	8.052026961

Confidence Level=99% (side channel)	
Confidence Level(99.0%)	0.104714876
Lower bound mean	15.86955898
Upper bound mean	16.07898874

Appendix E

Calculated Evaluation Metric Results for Scenario 5Node@16kB/s with 15.86% FE (FTP data frames from port 51434)

Threshold	True(+)	False(+)	True(-)	False(-)	Accuracy	Sensitivity	Specificity	Precision	F-Score
1	6413	4160	22602	0	0.8746	1.0000	0.8446	0.6065	0.7551
2	6413	4137	22625	0	0.8753	1.0000	0.8454	0.6079	0.7561
3	6413	4037	22725	0	0.8783	1.0000	0.8492	0.6137	0.7606
4	6413	3863	22899	0	0.8836	1.0000	0.8557	0.6241	0.7685
5	6413	3457	23305	0	0.8958	1.0000	0.8708	0.6497	0.7877
6	6413	2943	23819	0	0.9113	1.0000	0.8900	0.6854	0.8134
7	6408	2316	24446	5	0.9300	0.9992	0.9135	0.7345	0.8467
8	6393	1673	25089	20	0.9490	0.9969	0.9375	0.7926	0.8831
9	6351	1090	25672	62	0.9653	0.9903	0.9593	0.8535	0.9168
10	6257	662	26100	156	0.9753	0.9757	0.9753	0.9043	0.9386
11	**6062**	**347**	**26415**	**351**	**0.9790**	**0.9453**	**0.9870**	**0.9459**	**0.9456**
12	5715	162	26600	698	0.9741	0.8912	0.9939	0.9724	0.9300
13	5199	64	26698	1214	0.9615	0.8107	0.9976	0.9878	0.8905
14	4493	26	26736	1920	0.9413	0.7006	0.9990	0.9942	0.8220
15	3647	8	26754	2766	0.9164	0.5687	0.9997	0.9978	0.7245
16	2752	3	26759	3661	0.8896	0.4291	0.9999	0.9989	0.6003
17	1913	1	26761	4500	0.8643	0.2983	1.0000	0.9995	0.4595
18	1233	0	26762	5180	0.8439	0.1923	1.0000	1.0000	0.3225
19	701	0	26762	5712	0.8278	0.1093	1.0000	1.0000	0.1971
20	323	0	26762	6090	0.8164	0.0504	1.0000	1	0.0959
21	155	0	26762	6258	0.8114	0.0242	1.0000	1	0.0472